Xilinx FPGA 实验指导书

目录

[1. Basys3硬件电路 0](#_Toc450210943)

[2. Verilog简明教程 9](#_Toc450210944)

[3. 数码管显示实验 11](#_Toc450210945)

[4. UART通讯实验 31](#_Toc450210946)

[5. 数字频率计实验 41](#_Toc450210947)

# Basys3硬件电路

Basys3是围绕着一个Xilinx Artix-7 FPGA芯片XC7A35T-1CPG236C搭建的，它提供了完整、随时可以使用的硬件平台，并且它适合于从基本逻辑器件到复杂控制器件的各种主机电路。Basys3上集成了大量的I/O设备和FPGA所需的电路，由此可以构建无数设备而不需其他器件。

主要规格/特殊功能

产品规格：

Basys3为想要学FPGA和数字电路设计的用户提供了一个理想的电路设计平台。Basys3开发板提供完整的硬件存取电路，可以完成从基本逻辑到复杂控制器的设计。四个标准扩展连接器配合用户设计的电路板，或Pmods（Digilent设计的A/D和D/A转换，电机驱动器，传感器输入等）其他功能。附带USB电缆提供电源和编程接口，使得其成为入门复杂数字电路设计系统的完美低成本平台。

关键特性：

1、 33,280 个逻辑单元，六输入LUT结构

2、 1,800 Kbits 快速RAM块

3、 5个时钟管理单元, 均各含一个锁相环 (PLL)

4、 90个DSP slices

5、 内部时钟最高可达450MHz

6、 1个片上模数转换器 (XADC)

外围设备：

1、 16个拨键开关

2、 16个LED

3、 5个按键开关

4、 4位7段数码管

5、 3个Pmod连接口

6、 一个专用AD信号Pmod接口

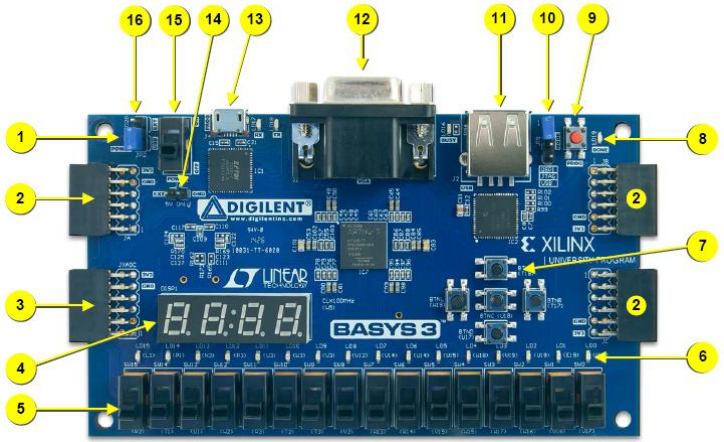
7、 12位的VGA输出接口

8、 USB-UART桥

9、 串口flash

10、用于FPGA编程和通信的USB-JTAG口

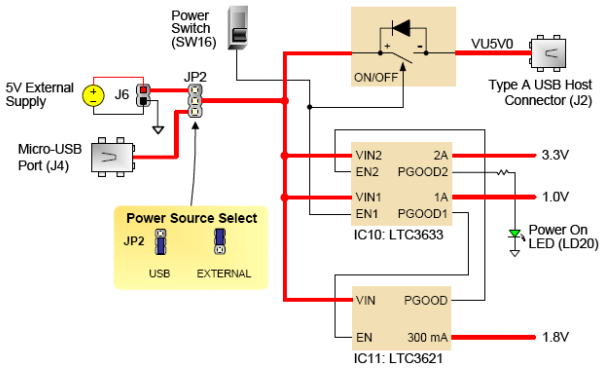
11、可连接鼠标、键盘、记忆棒的USB口





* 1. 电源电路

Basys3开发板可以通过2种方式进行供电，一种是通过J4的USB 端口供电；另一种是通过J6的接线柱进行供电（5V）。通过JP2跳线帽的不同选择进行供电方式的选择。电源开关通过SW16进行控制，LD20为电源开关的指示灯。电源的电路如下图所示：

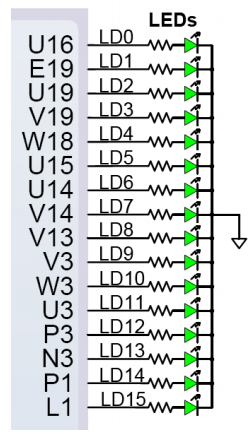


说明，如果选用外部电源（即J6）那么应该保证：1，电源电压在4.5V-5.5V 范围内；2，至少能提供1A的电流。

注意：只有在特别情况下电源电压才可以使用3.6V电压。

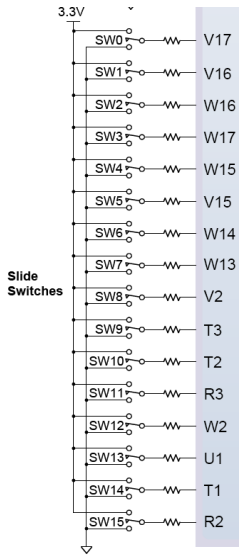
* 1. LED灯电路

LED部分的电路如图所示。当FPGA输出为高电平时，相应的LED点亮；否则，LED 熄灭。板上配有16个LED，在实验中灵活应用，可用作标志显示或代码调试的结果显示，既直观明了又简单方便。



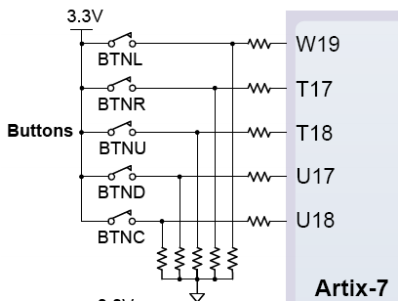
1.3 拨码开关电路

拨码开关的电路如图所示。在使用这个16位拨码开关时请注意一点，当开关打到下档时，表示FPGA的输入为低电平：



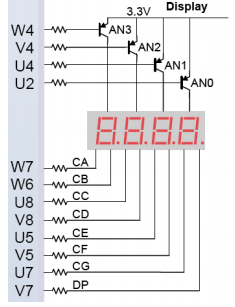
1.4 按键电路

按键部分的电路如图所示。板上配有5个按键，当按键按下时，表示FPGA的相应输入脚为高电平。在学习过程中，建议每个工程都有一个复位输入，这对代码调试将大有好处：



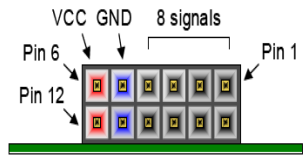
1.5 数码管电路

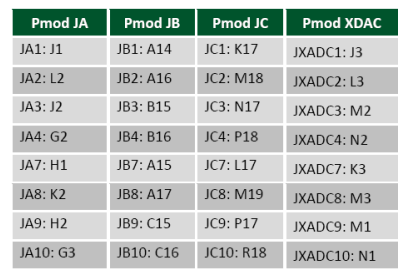
数码管显示部分的电路如图所示。我们使用的是一个四位带小数点的七段共阳数码管，当我们相应的输出脚为低电平时，该段位的LED点亮。位选位也是低电平选通：



1.6 IO扩展电路

4个标准的扩展连接器（其中一个为专用AD信号Pmod接口）允许设计使用面包板、用户设计的电路或Pmods扩展Basys3板，（Pmods是价格便宜的模拟和数字I / O模块，能提供一个A/ D＆D / A 转换，电机驱动器，传感器投入和许多其他功能）。



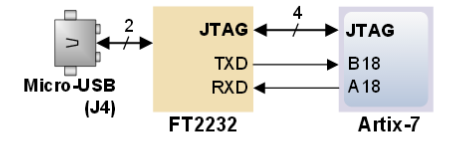


1.7 USB—UART转换电路

Basys3开发板包含由FT2232芯片搭建而成的USB—UART的转换电路。这使得你可以通过标准的Windows系统的COM端口用PC机上的应用于开发板进行通讯。在安装好驱动后，PC机可以从COM口向FPGA的B18和A18引脚发送I/O命令，从而产生串行数据通信。

Basys3开发板上提供两个状态指示灯，来提供产生数据通信时的直观视觉提示：发送数据指示灯LED（LD18）与接收数据指示灯（LD17）。

FT2232芯片同时可以用来控制USB—JTAG电路，然而两个电路的功能可以独立运行，并且两个电路同时运行时并不会相互影响。因此虽然只有一根Micro USB线，却可以同时进行程序下载、UART通讯以及供电。



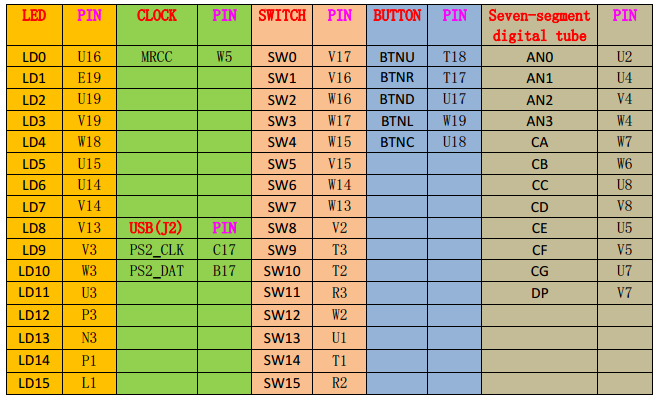
1.8 FPGA调试及配置电路

上电后，Basys3板上必须配置FPGA，然后才能执行任何有用功能。在配置过程中，bit文件转移到FPGA内存单元中实现逻辑功能和电路互连。通过赛灵思公司免费的Vivado软件可以通过VHDL，Verilog语言，或基于原理图的源文件创建.bit文件。

下载程序有3种方式：

1. 用Vivado通过JTAG方式下载.bit文件到FPGA芯片。
2. 用Vivado通过QSPI方式下载.bit文件到Flash芯片，实现掉电不易失。
3. 用U盘或移动硬盘通过J2的USB端口下载.bit文件到FPGA芯片（建议将.bit文件放到U盘根目录下，且只放1个），该U盘应该是FAT32文件系统。

1.9 引脚分配表格





# Verilog简明教程

Verilog HDL是一种硬件描述语言，用于从算法级、门级到开关级的多种抽象设计层次的数字系统建模。被建模的数字系统对象的复杂性可以介于简单的门和完整的电子数字系统之间。数字系统能够按层次描述，并可在相同描述中显式地进行时序建模。

模块是Verilog的基本描述单位，用于描述某个设计的功能或结构及其与其他模块通信的外部端口。一个设计的结构可使用开关级原语、门级原语和用户定义的原语方式描述; 设计的数据流行为使用连续赋值语句进行描述; 时序行为使用过程结构描述。一个模块可以在另一个模块中调用。一个模块的基本语法如下：

module module\_name (port\_list);

Declarations:

reg, wire, parameter,

input, output, inout, function, task, . . .

Statements:

Initial statement

Always statement

Module instantiation

Gate instantiation

UDP instantiation

Continuous assignment

endmodule

模块的定义从关键字module开始，到关键字endmodule结束，每条Verilog HDL语句以“；”做为结束（块语句、编译向导、endmodule等少数除外）。一个完整的Verilog模块由以下四个部分组成：

1．模块定义行：module module\_name (port\_list);

2．说明部分用于定义不同的项，例如模块描述中使用的寄存器和参数。语句定义设计的功能和结构。说明部分和语句可以散布在模块中的任何地方；但是变量、寄存器、 线网和参数等的说明部分必须在使用前出现。为了使模块描述清晰和具有良好的可读性, 最好将所有的说明部分放在语句前。说明部分包括：寄存器，线网，参数：reg, wire, parameter 端口类型说明行：input, output, inout函数、任务：function, task,等

3．描述体部分：这是一个模块最重要的部分，在这里描述模块的行为和功能，子模块的调用和连接，逻辑门的调用，用户自定义部件的调用，初始态赋值，always块，连续赋值语句等等。

4．结束行，以endmodule结束，注意后面没有分号了。

下例为always语句对1位全加器电路建模的示例。

module FA\_Seq (A, B, Cin, Sum, Cout);

input A, B, Cin;

output Sum, Cout;

reg Sum, Cout;

reg T1, T2, T3;

always @ ( A or B or Cin )

begin

Sum = (A ^ B) ^ Cin;

T1 = A & Cin;

T2 = B & Cin;

T3 = A & B;

Cout = (T1| T2) | T3;

end

endmodule

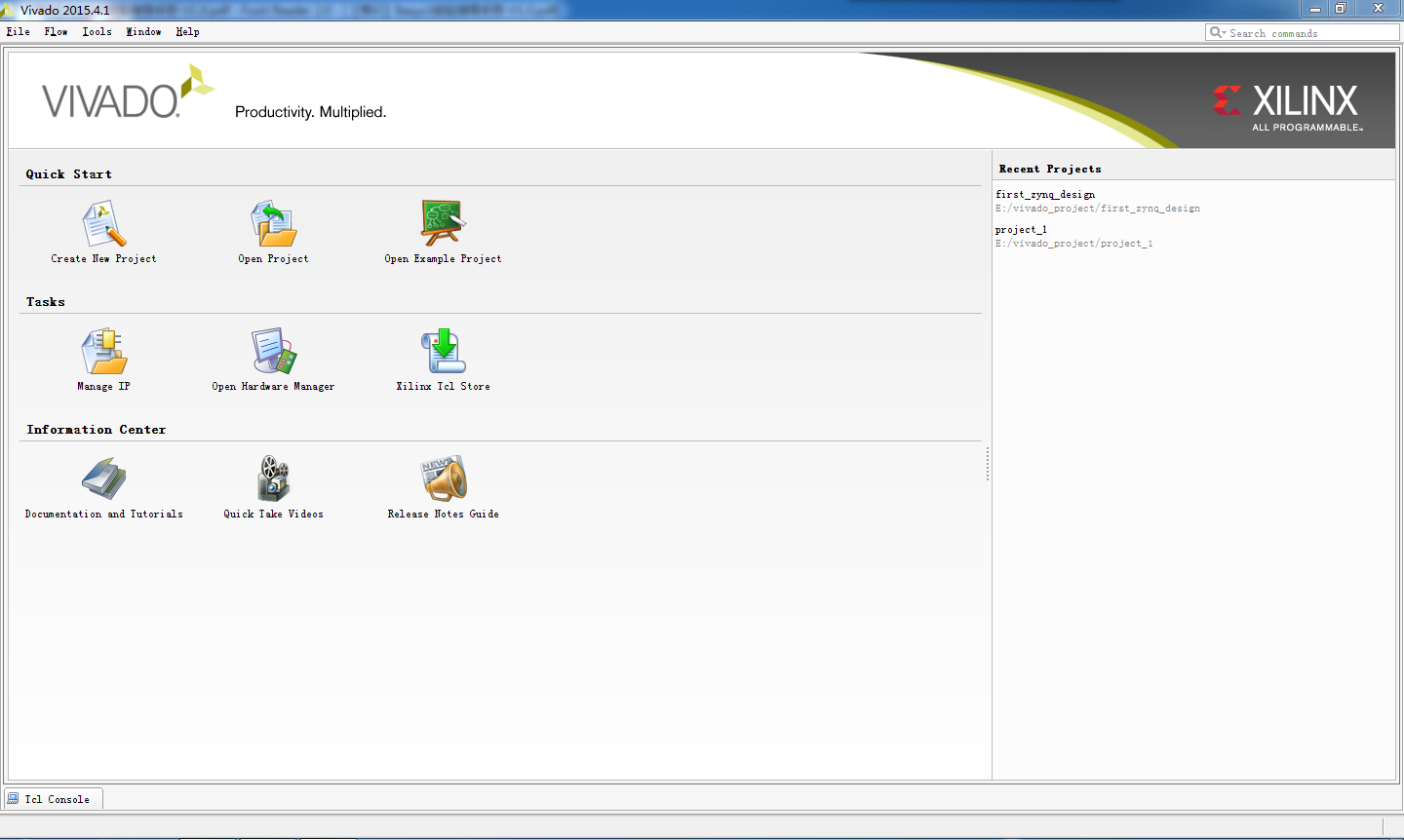
模块FA\_Seq有三个输入和两个输出。由于Sum、Cout、T1、T2和T3在always 语句中被赋值,它们被说明为 reg 类型(reg是寄存器数据类型的一种)。always语句中有一个与事件控制(紧跟在字符@后面的表达式)。相关联的顺序过程(begin-end对)。这意味着只要A、B或Cin上发生事件，即A、B或Cin之一的值发生变化，顺序过程就执行。在顺序过程中的语句顺序执行，并且在顺序过程执行结束后被挂起。顺序过程执行完成后，always 语句再次等待A、B或Cin上发生的事件。

# 数码管显示实验

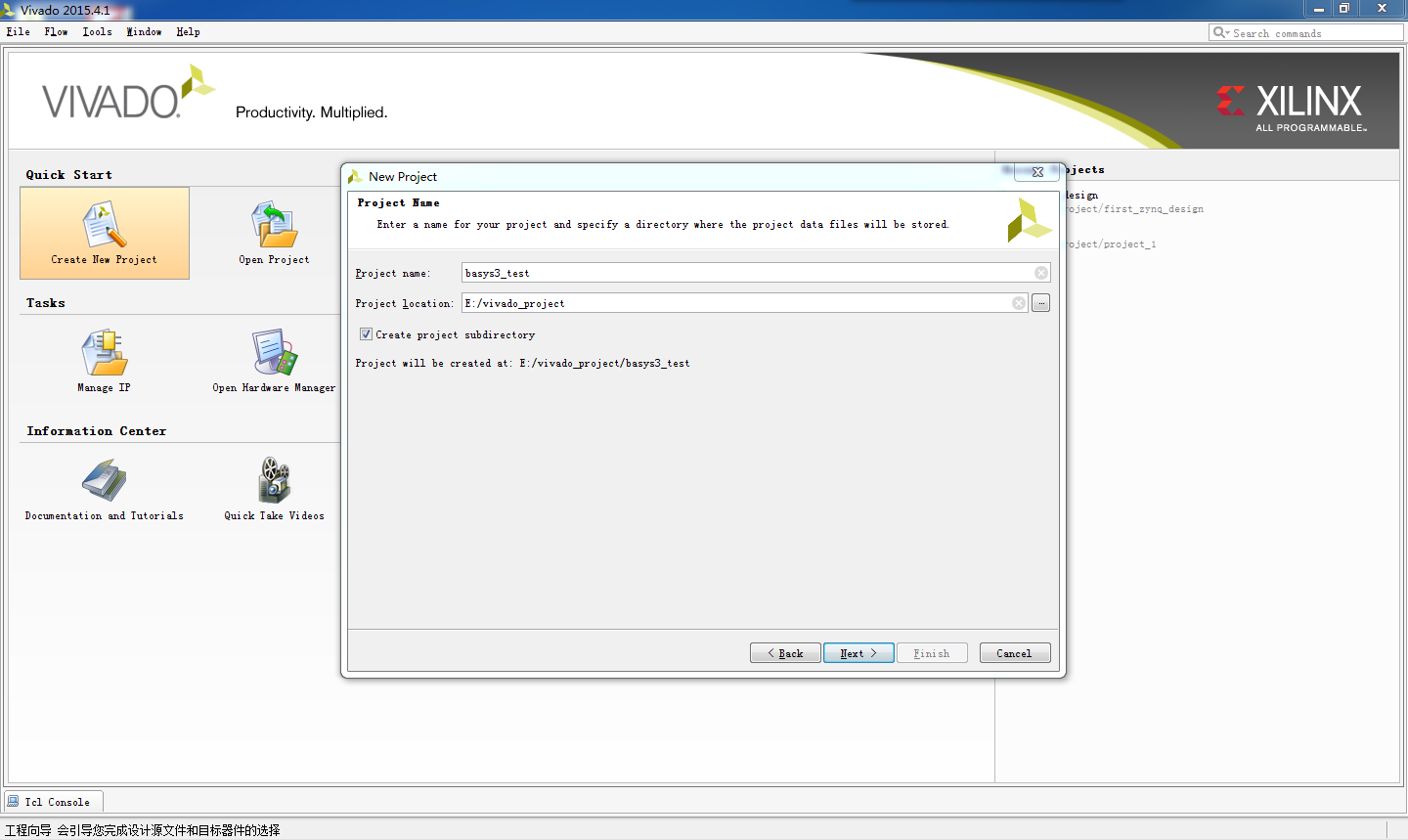
利用basys3开发板做最简单的一个实验。使开发板上的4个7段数码管显示数字“1234”或者“4321”。由拨动开关sw0来控制数字顺序或者逆序显示。具体操作过程如下

首先，需要创建一个项目工程。

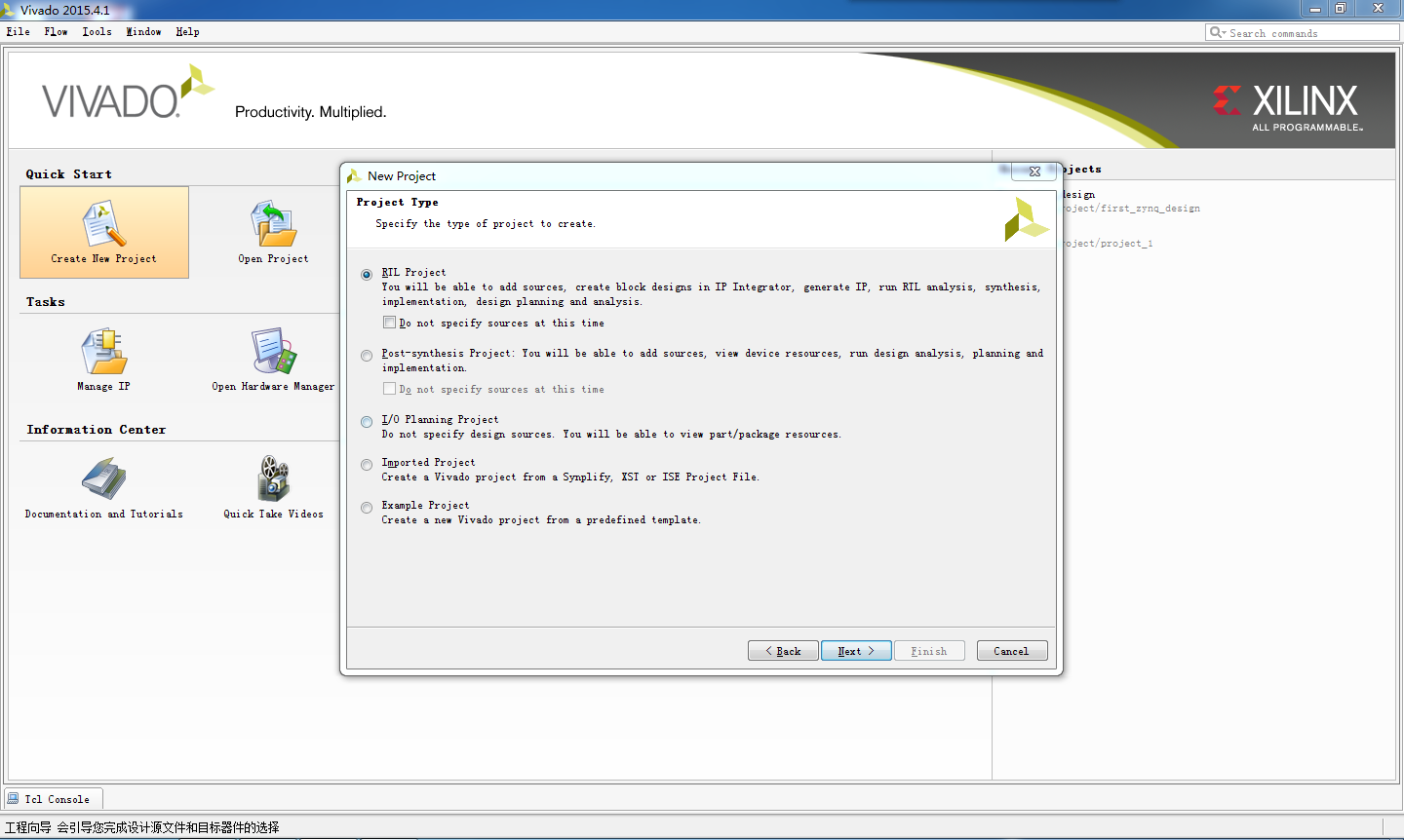
打开vivado，界面如下图，点击create new project



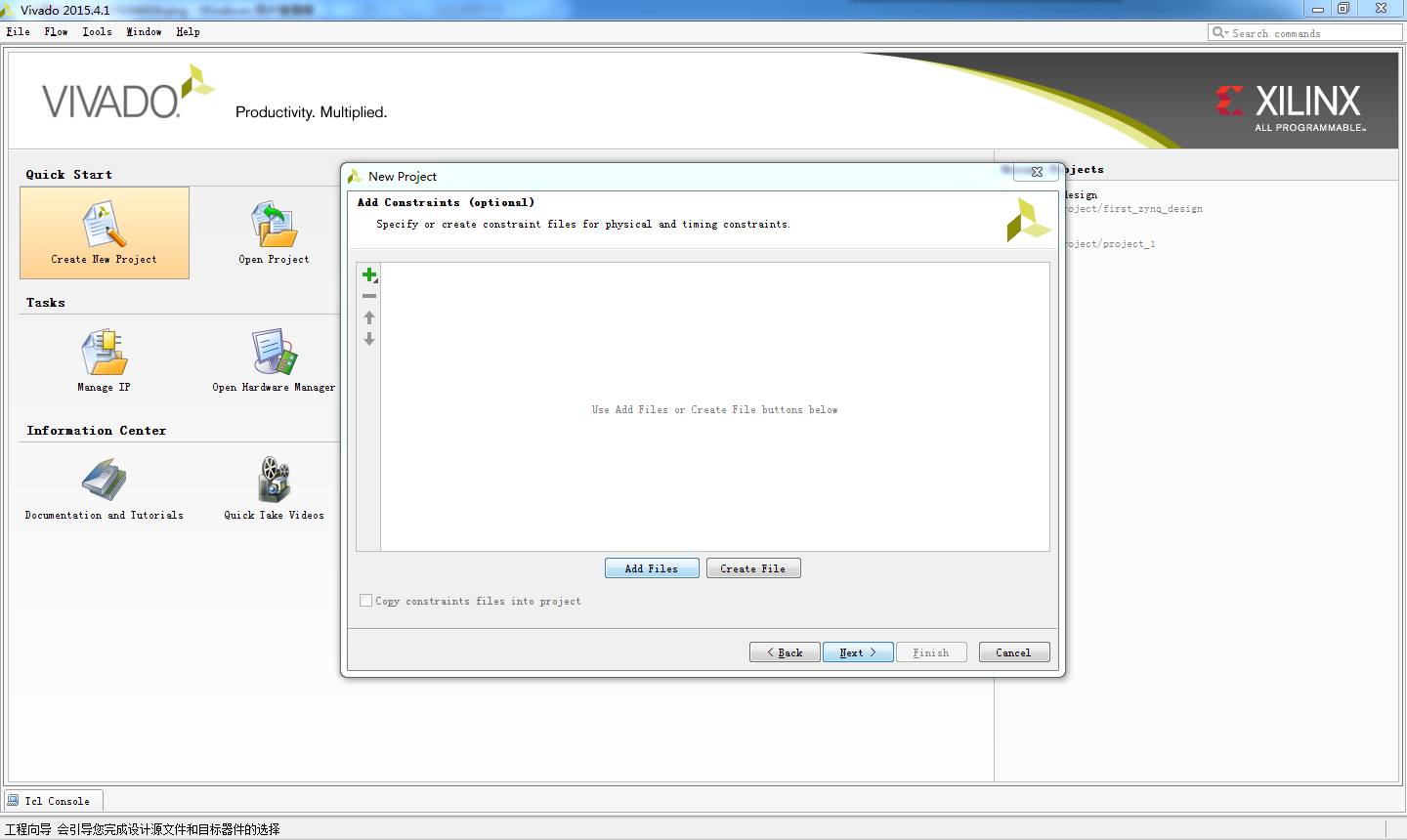
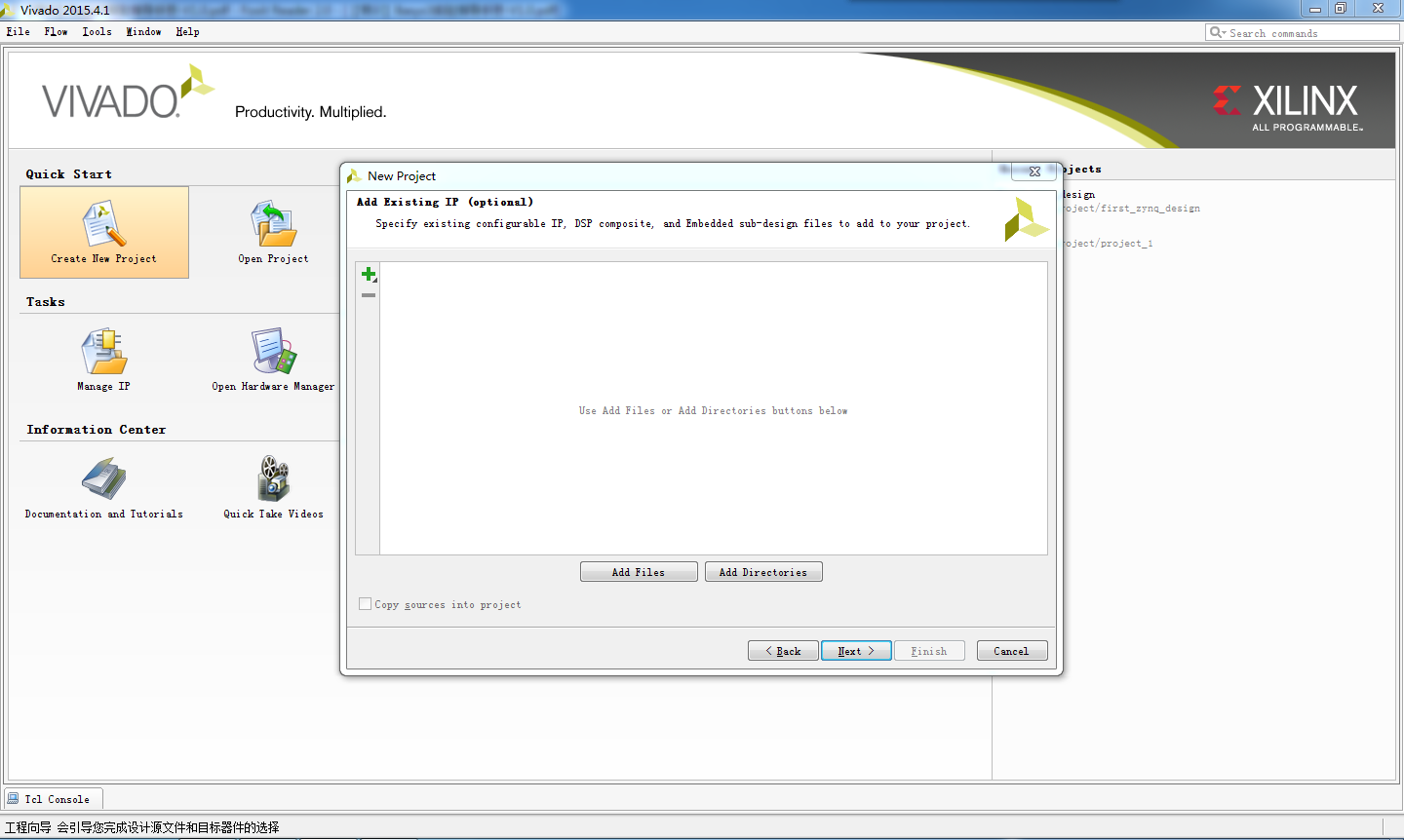
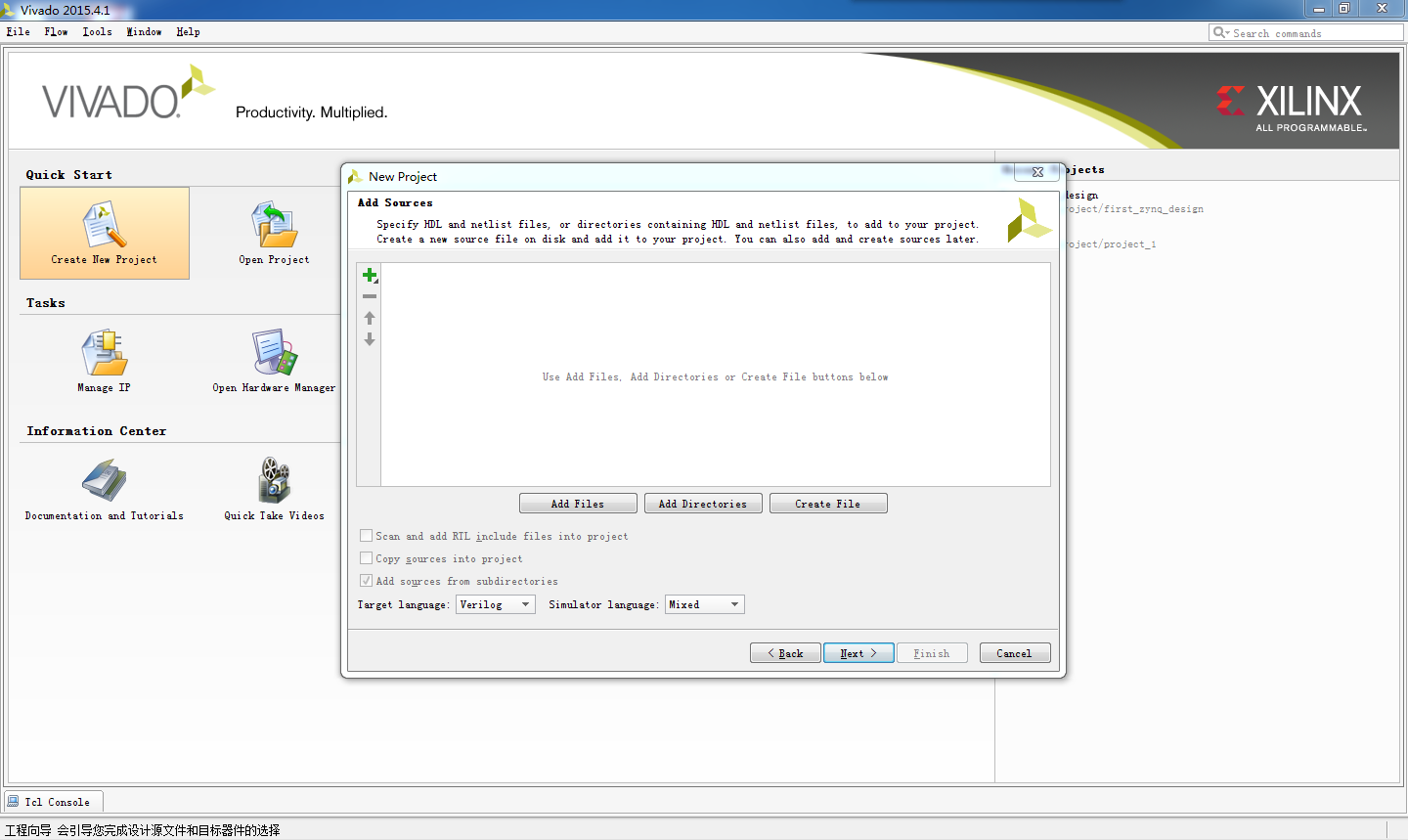
输入工程名称，选择存储位置，勾选create project subdirectory



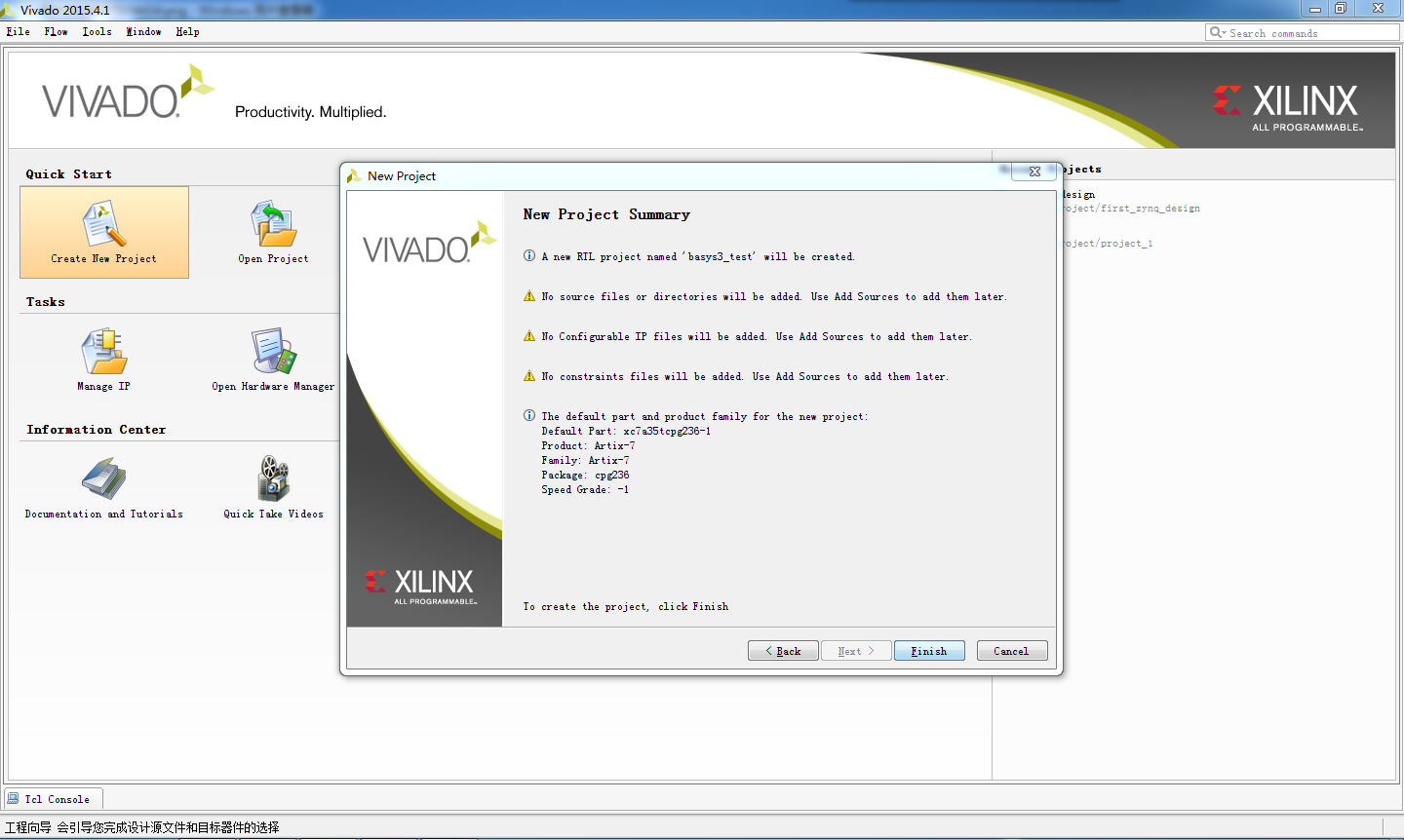
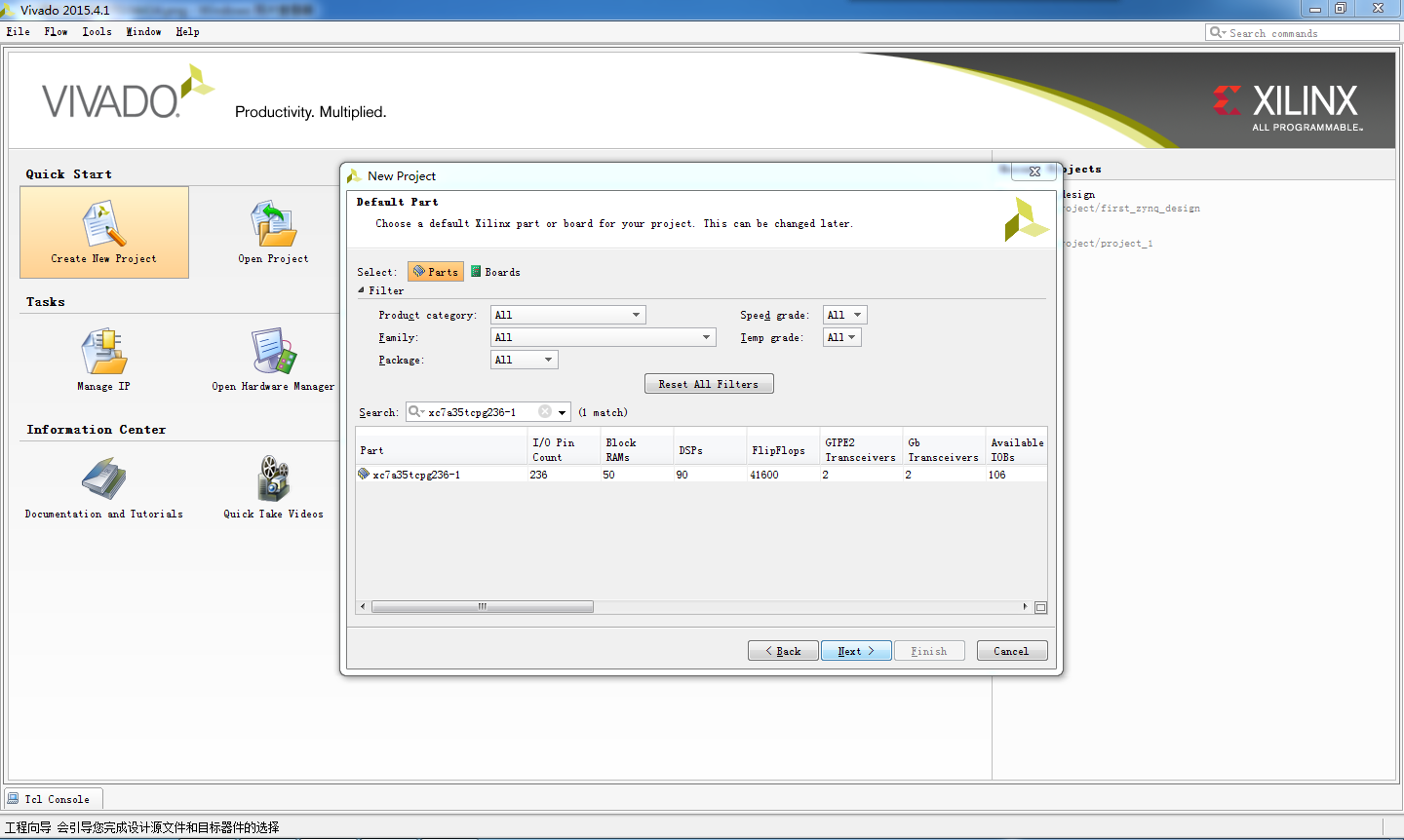
选择RTL project



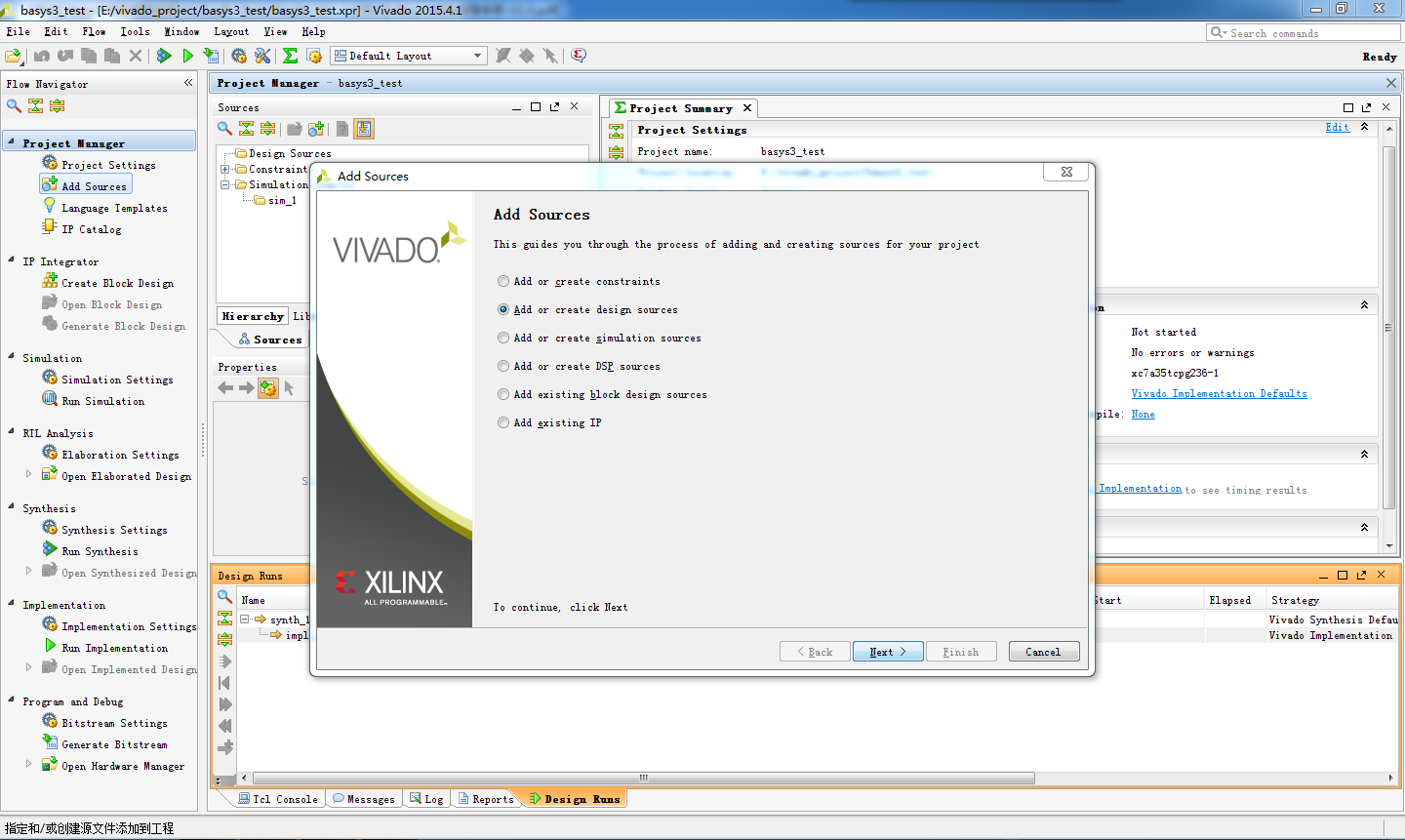
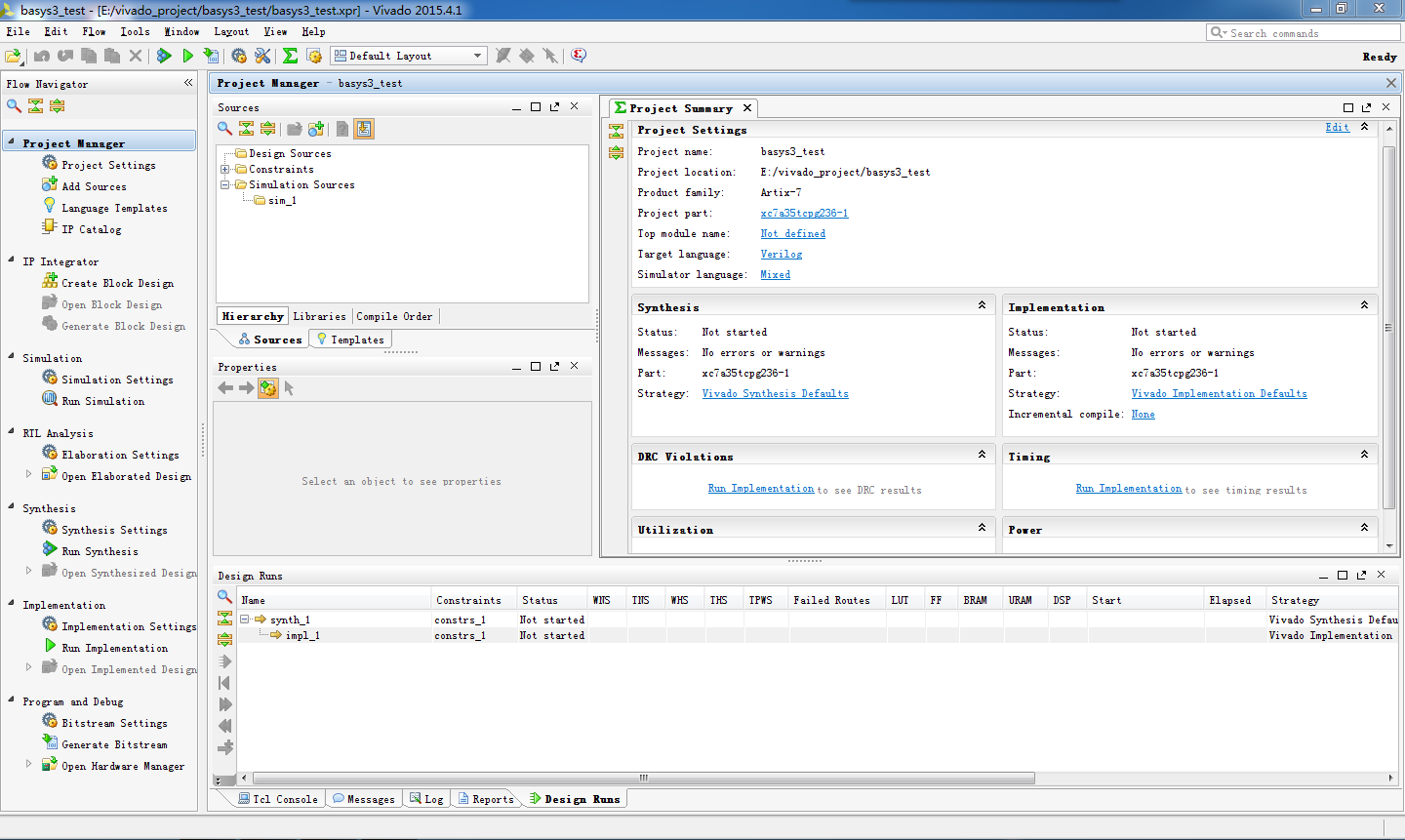
因为现在不需要添加source，IP以及constraints，所以接下来几步都直接点击next



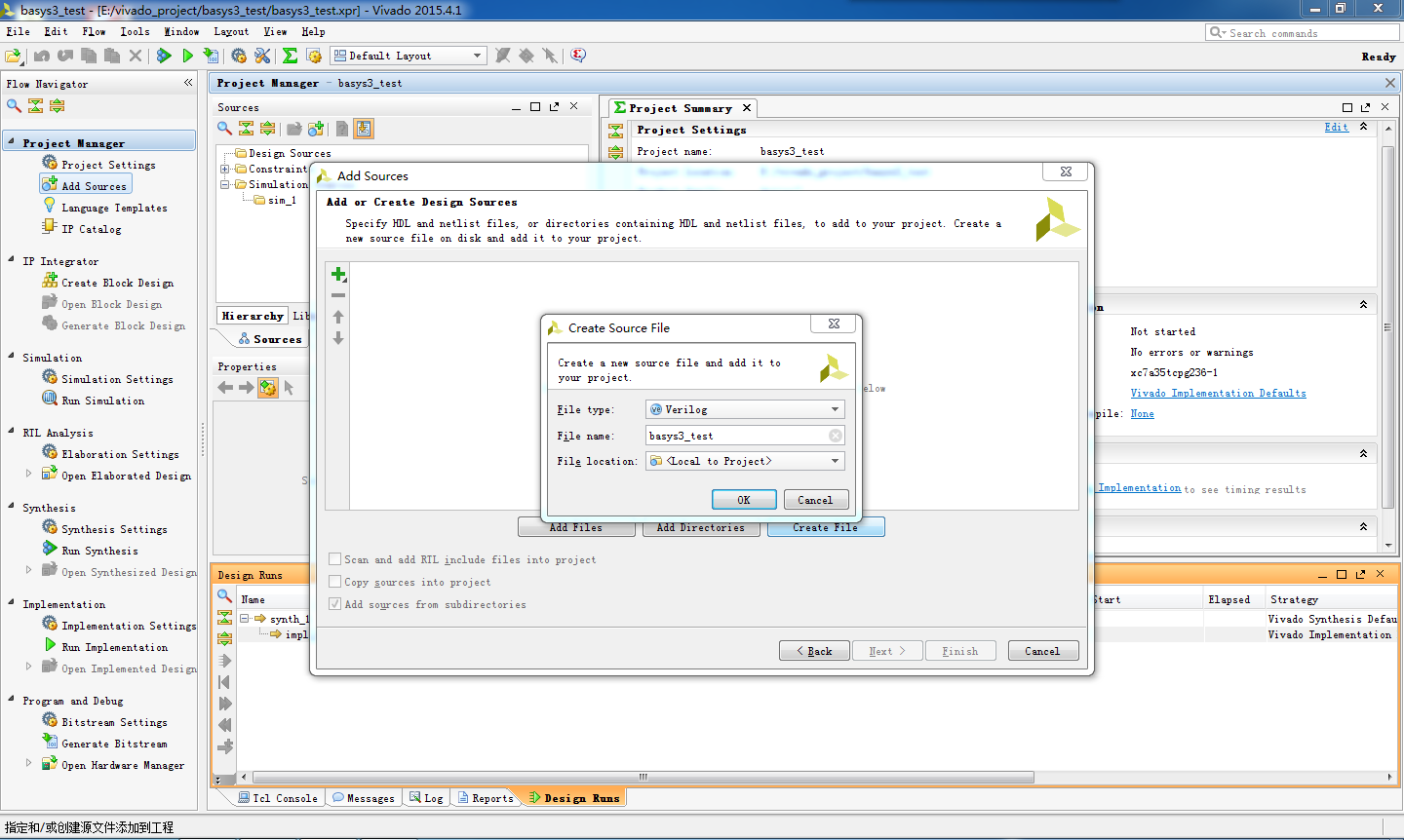
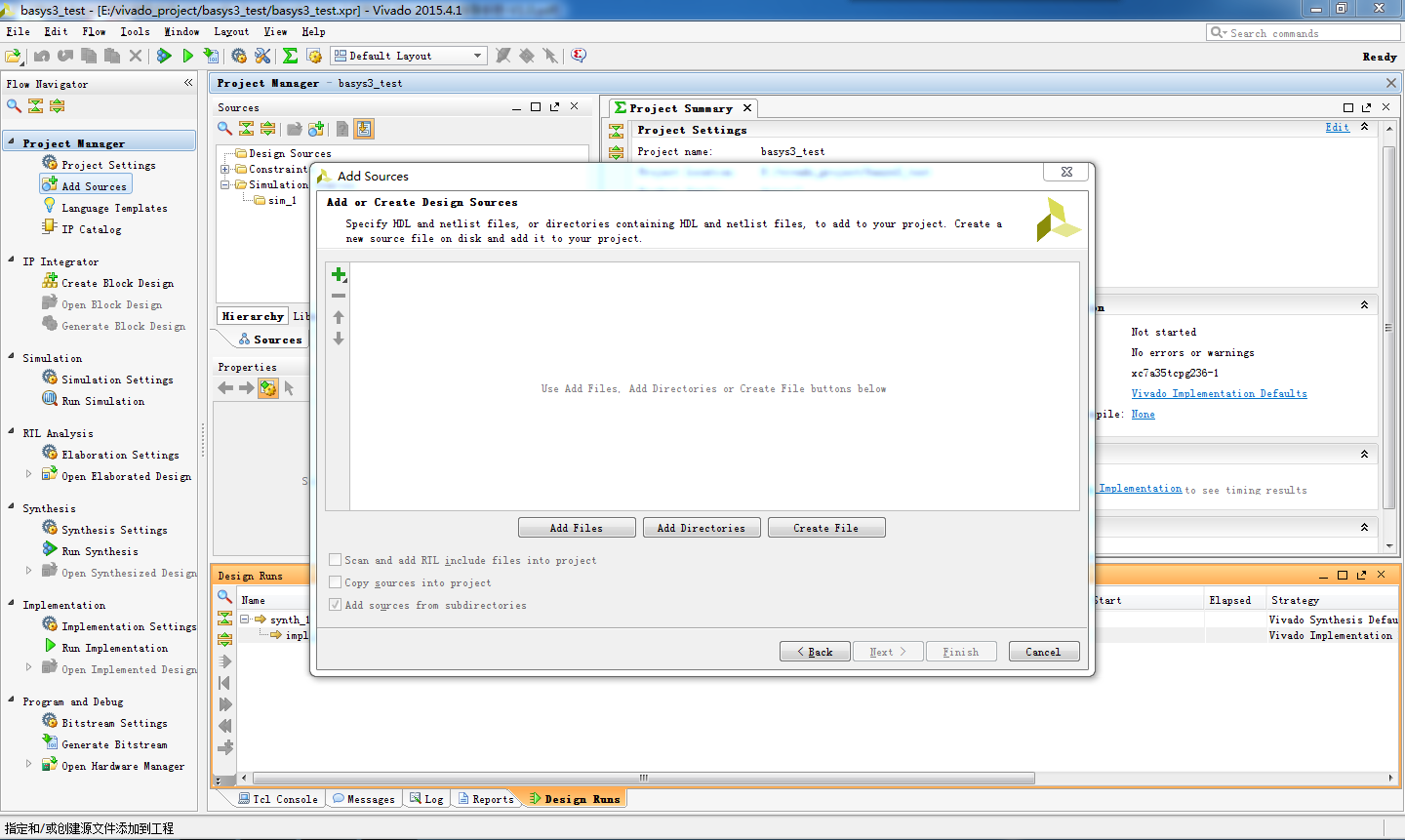
选择板子型号。单击parts，然后输入xc7a35tcpg236-1，这就是basys3的型号代码。选中筛选出的结果，然后finish



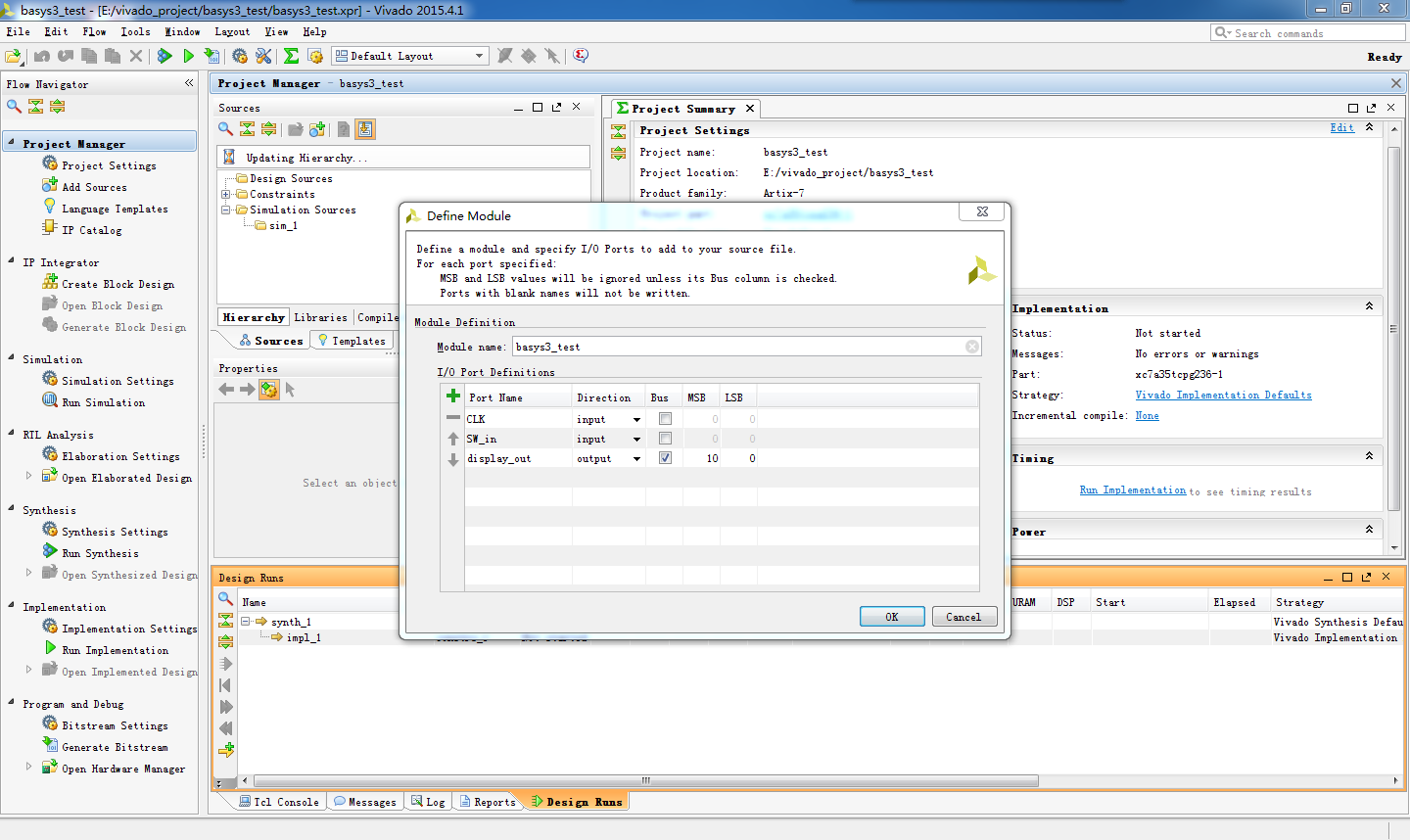
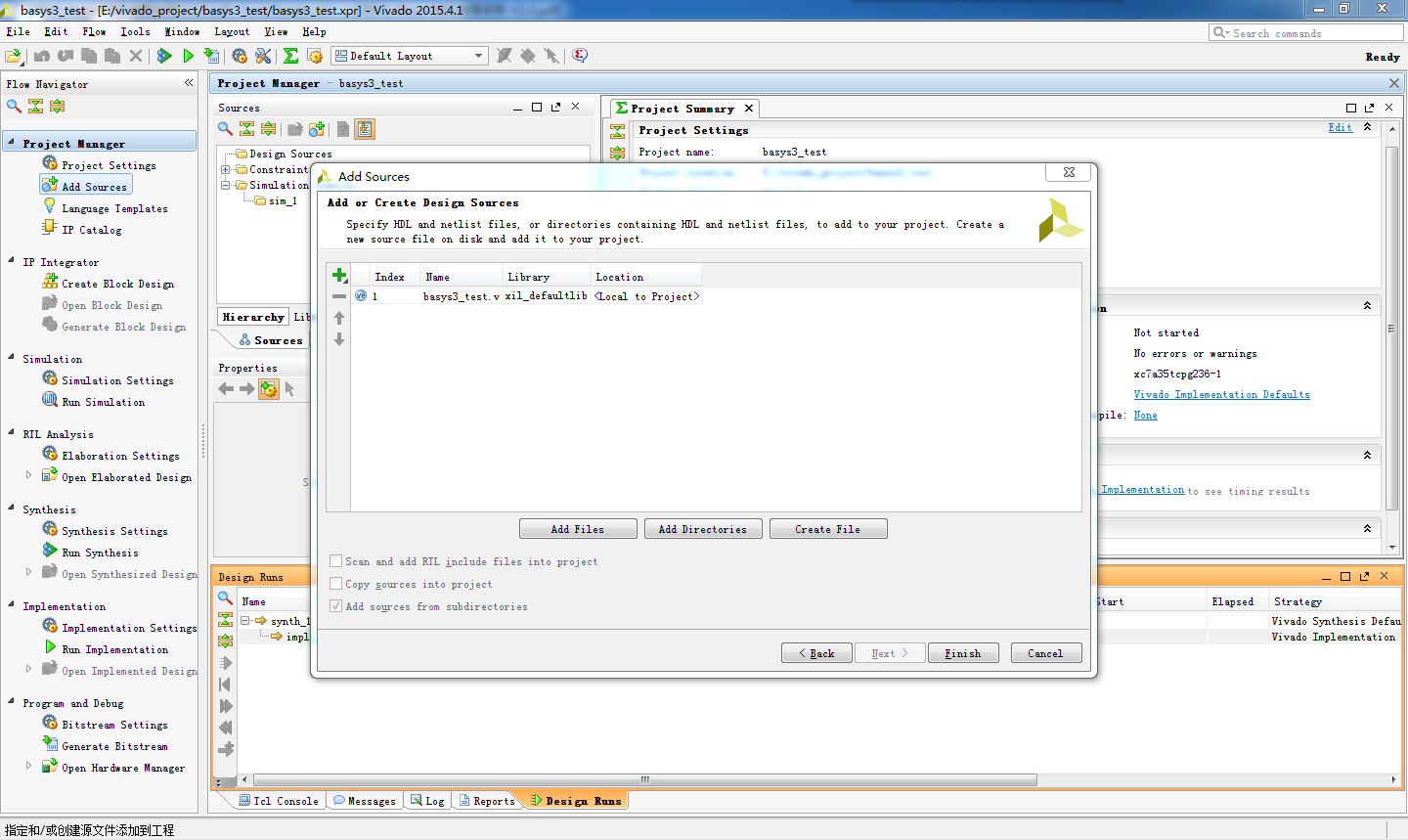
生成界面如图，点击左侧导航栏中的add source，然后选择add or create design sources，创建源文件



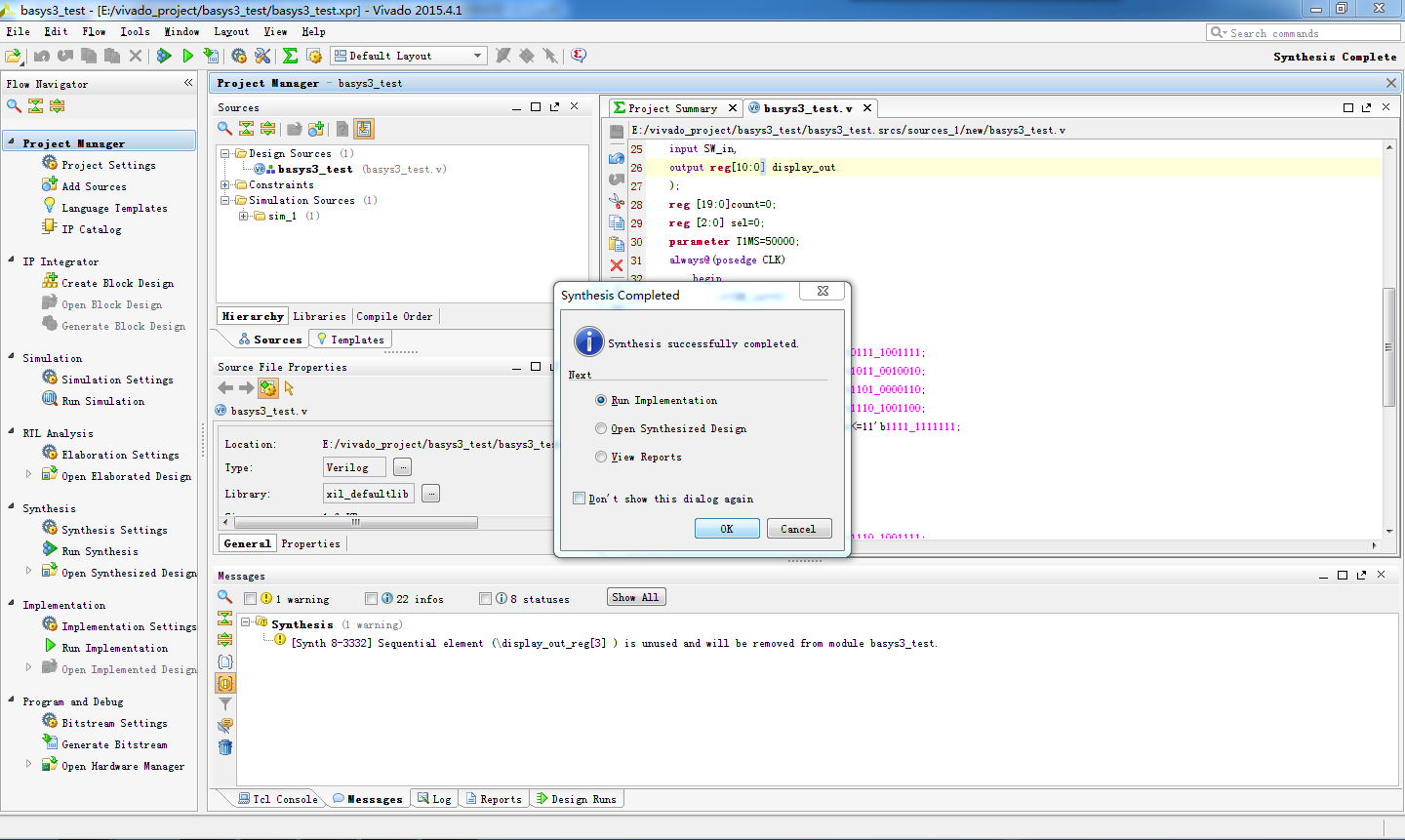
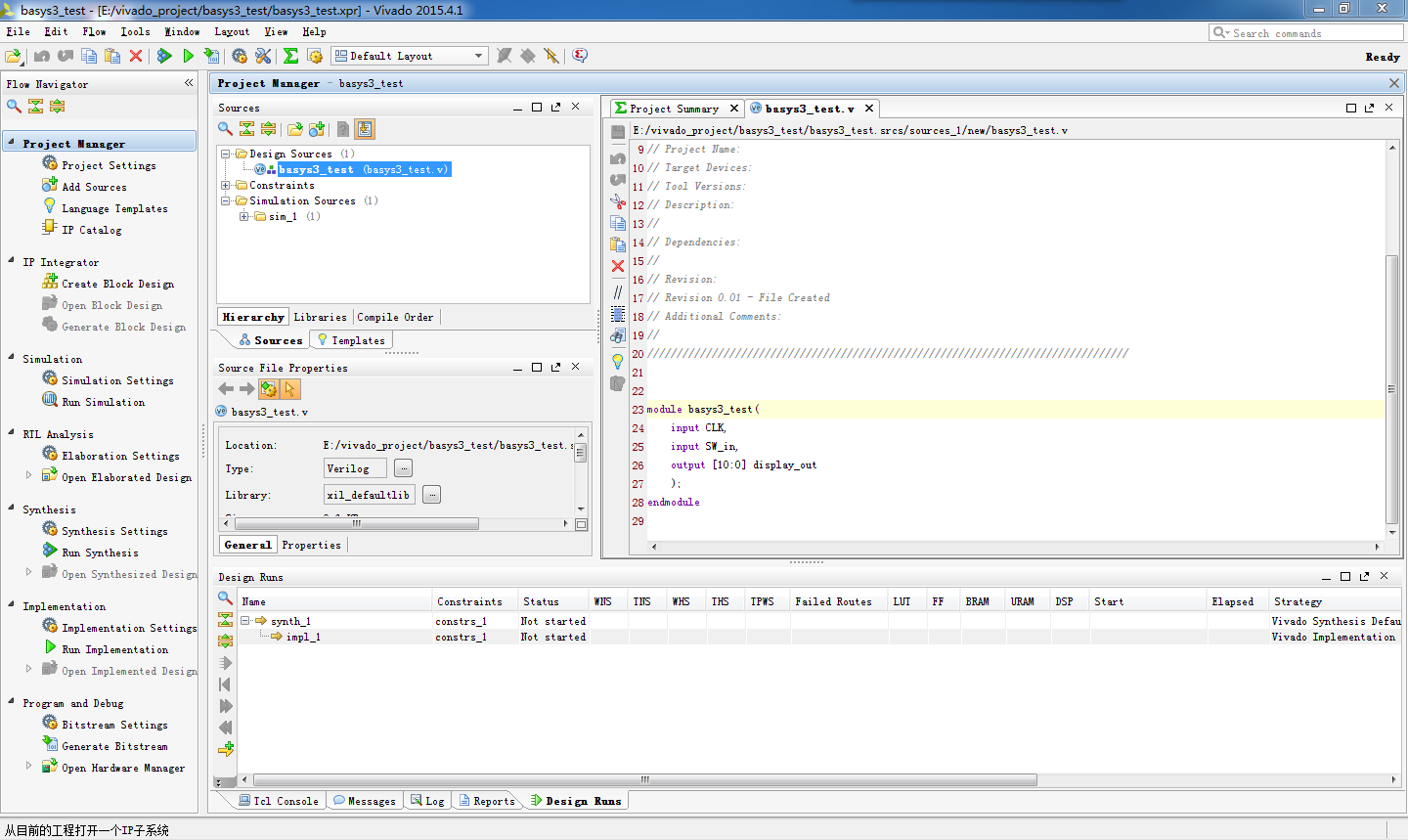
点击create file，然后选择语言Verilog，填入文件名，点击ok



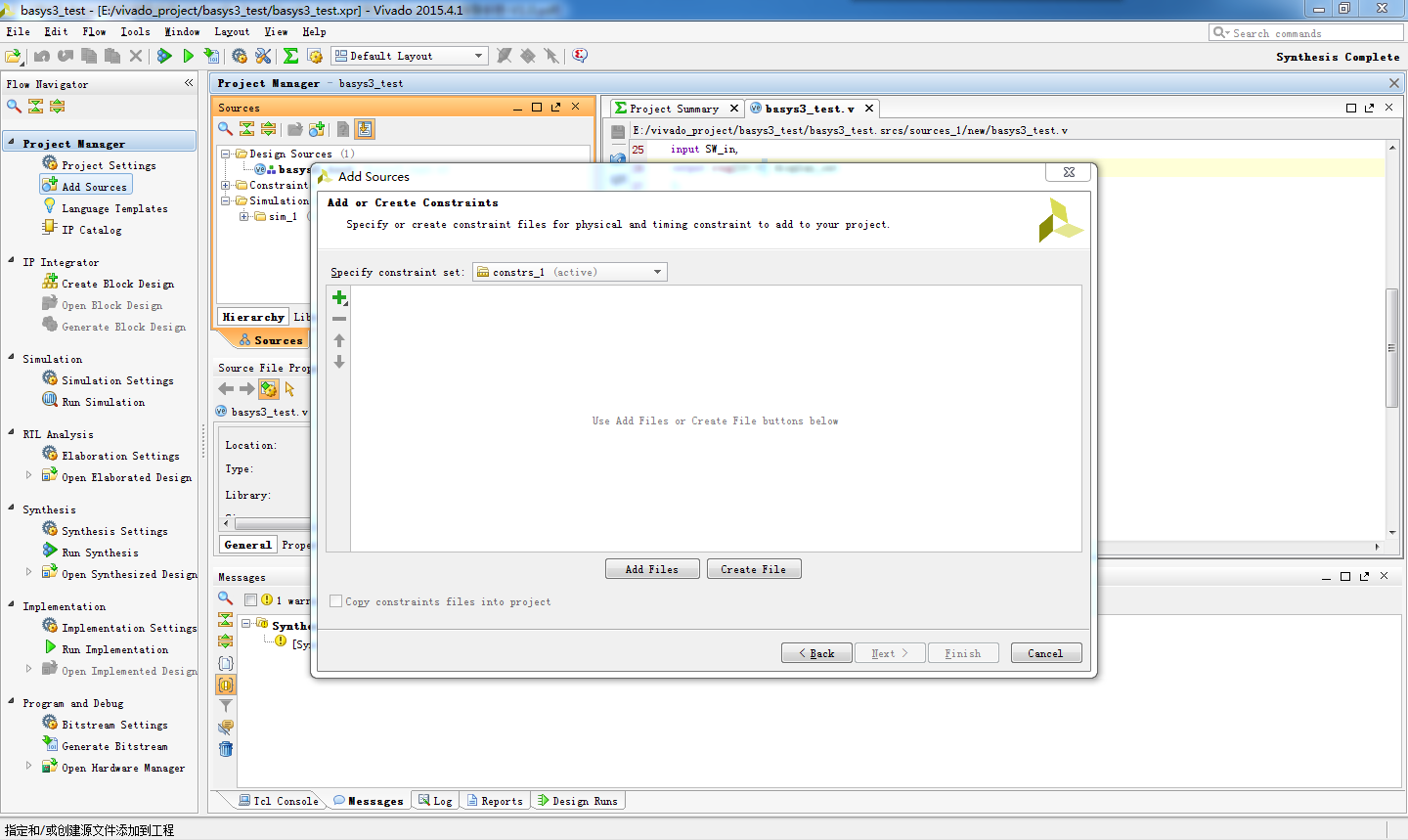
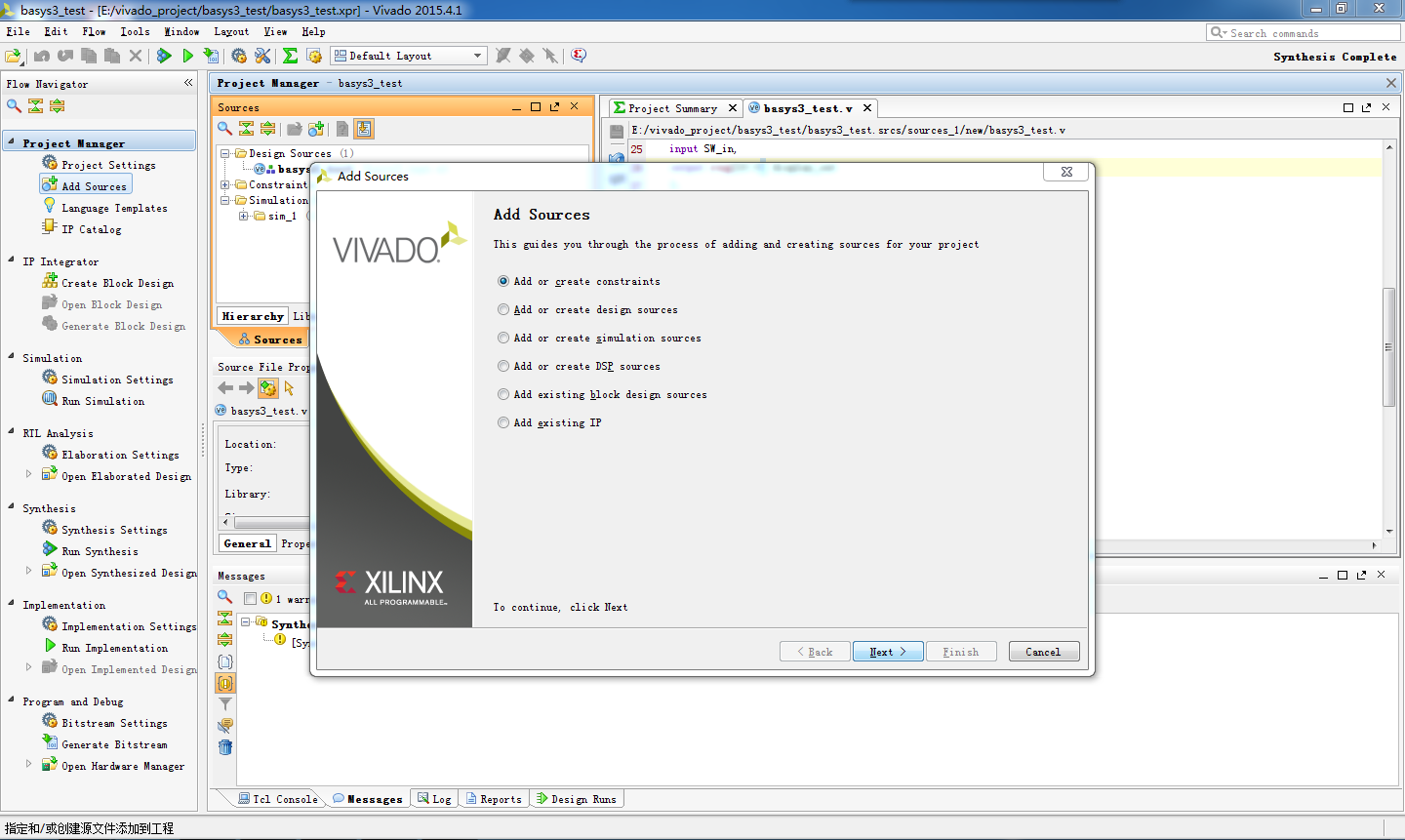
点击finish，然后如图定义模块的输入输出



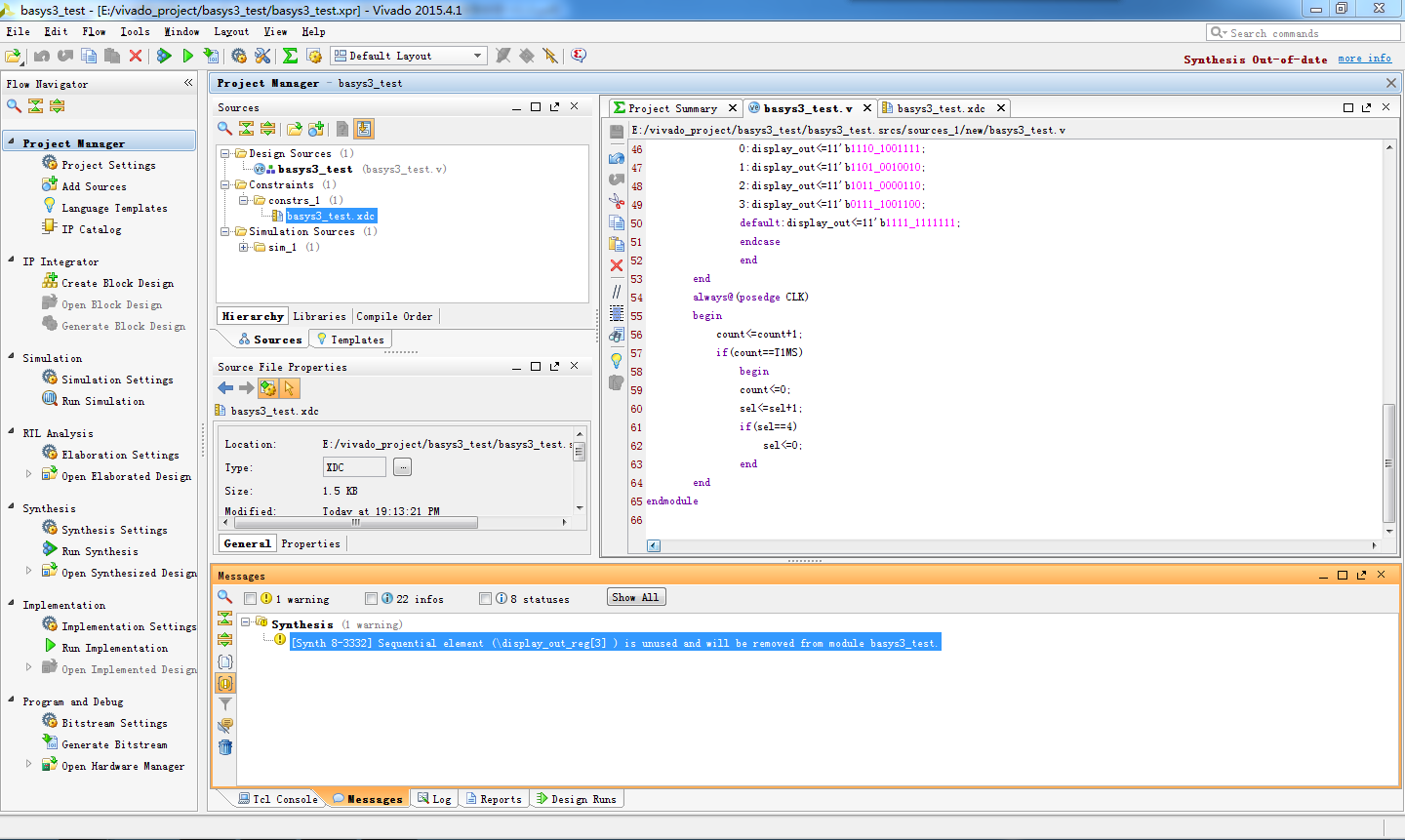
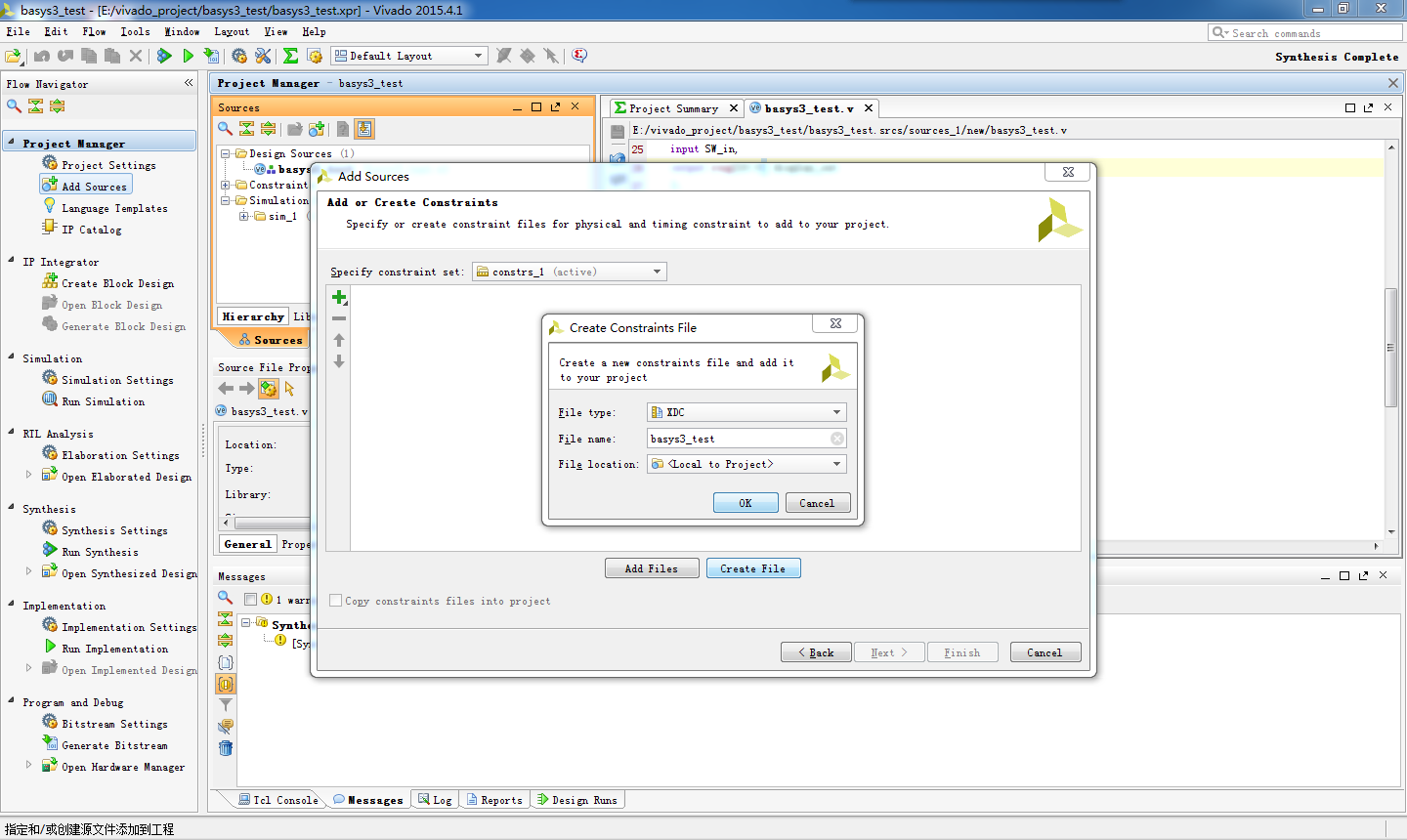
在此界面中双击源文件，然后在右侧工作区编写代码。（参考代码在文末）编写完成后点击左侧导航栏中的run synthesis进行综合。综合是针对给定的电路实现功能和实现电路的约束条件，如速度、功耗、成本以及电路的类型等，通过计算机进行优化处理，获得一个能满足上述要求或者相近的最优电路设计方案。综合完成后弹出如下窗口，先将其关闭



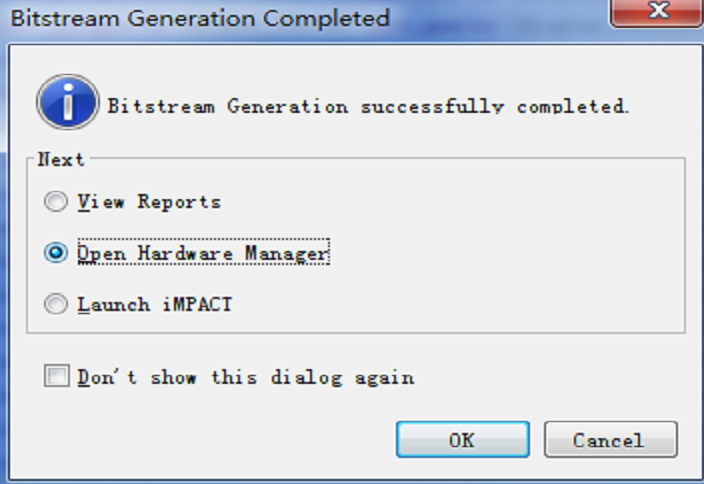
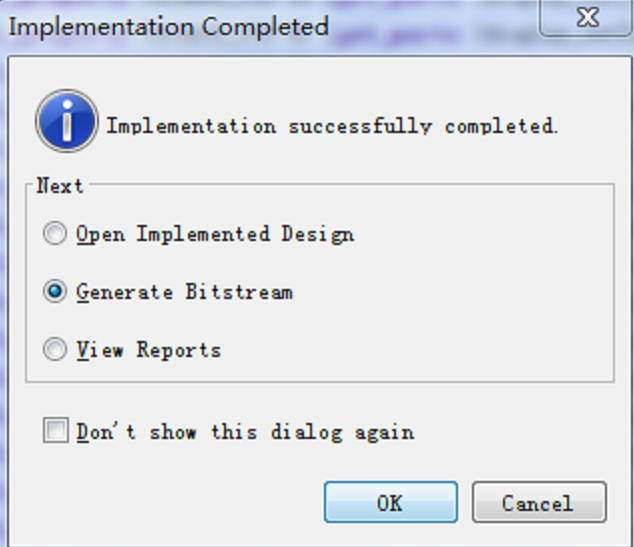
再次点击add sources，选择add or create constraints。创建约束文件，用以约束时序、引脚等



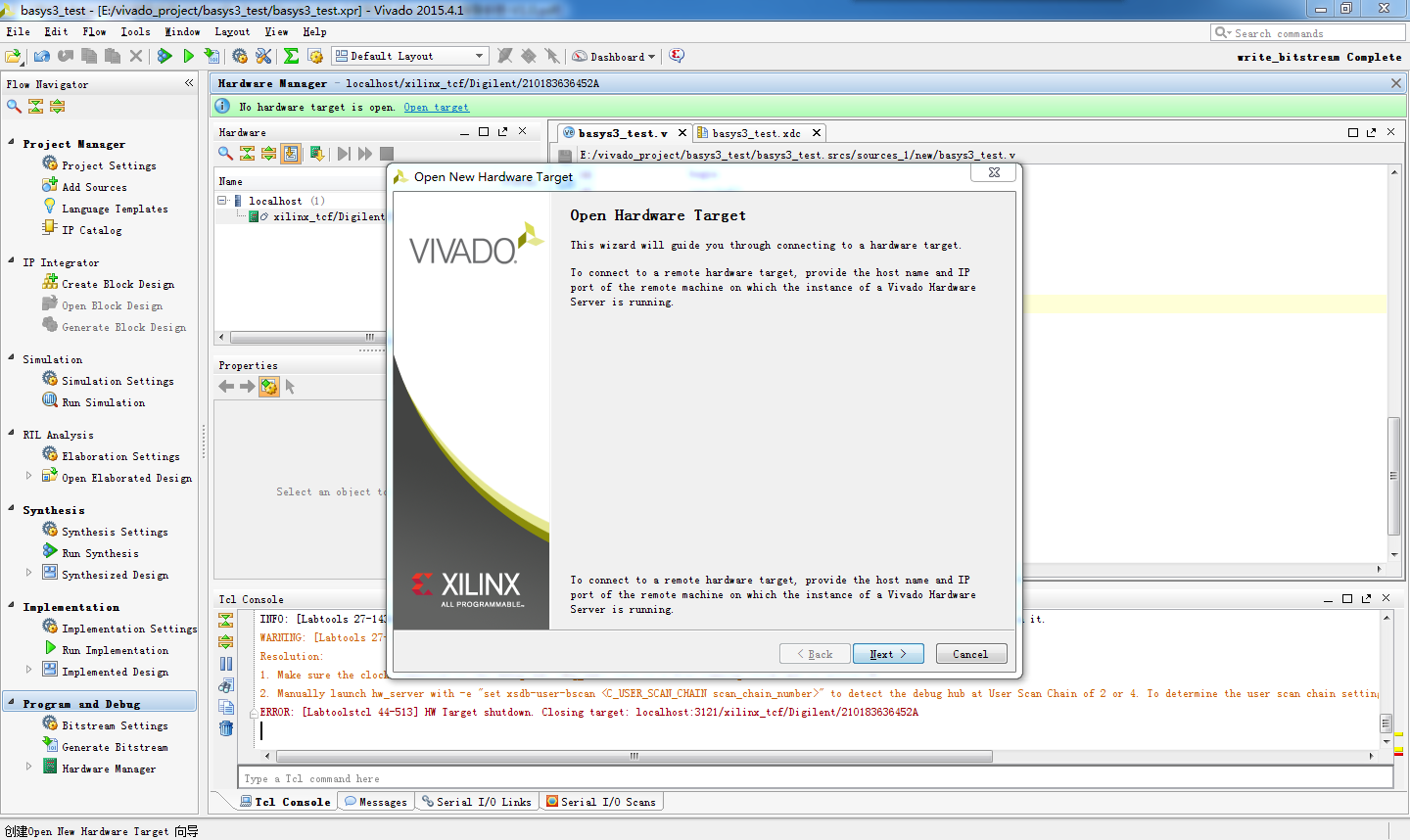
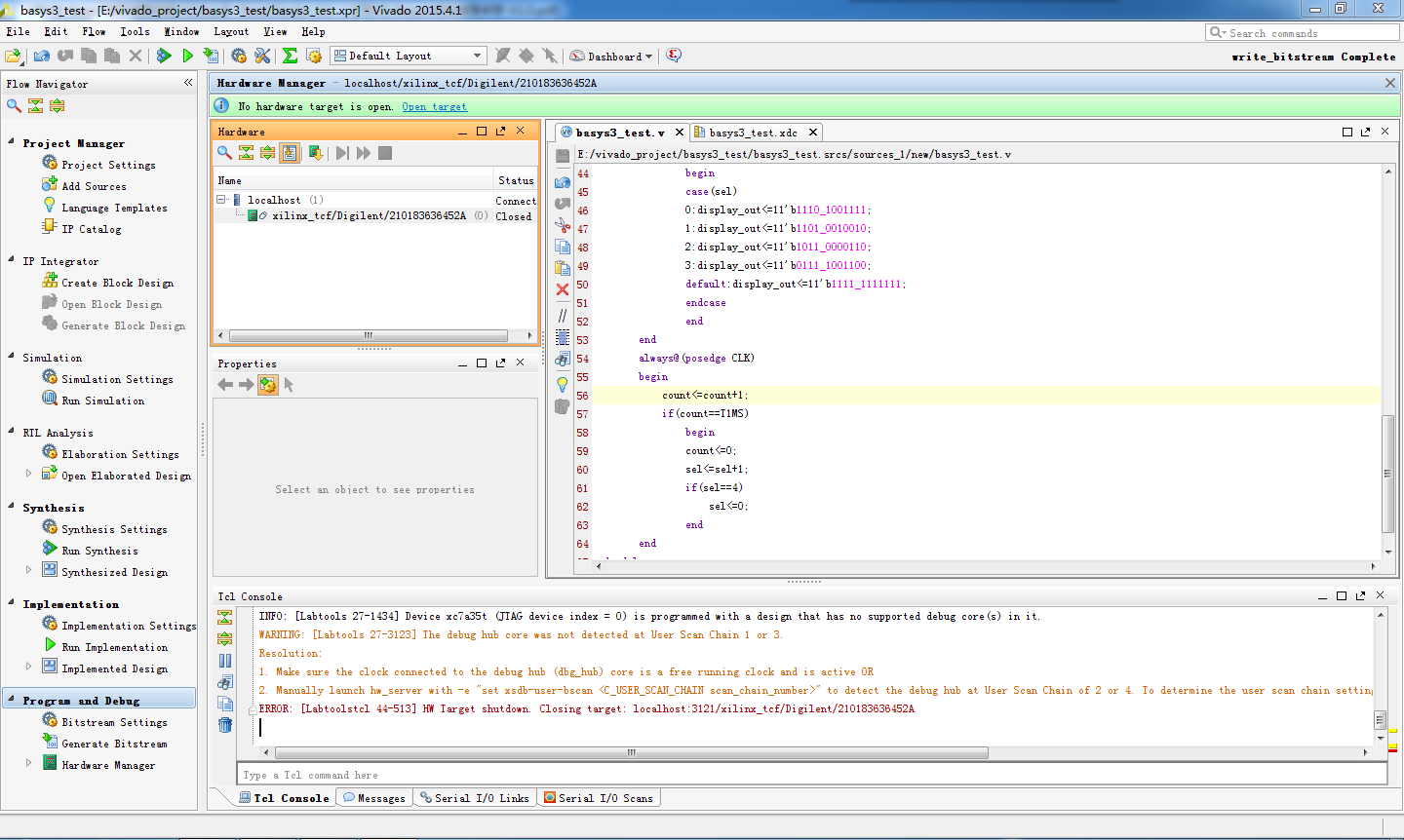
同样选择create file并填入文件名，然后双击创建的约束文件，在右侧工作区编写代码（参考代码在文末）



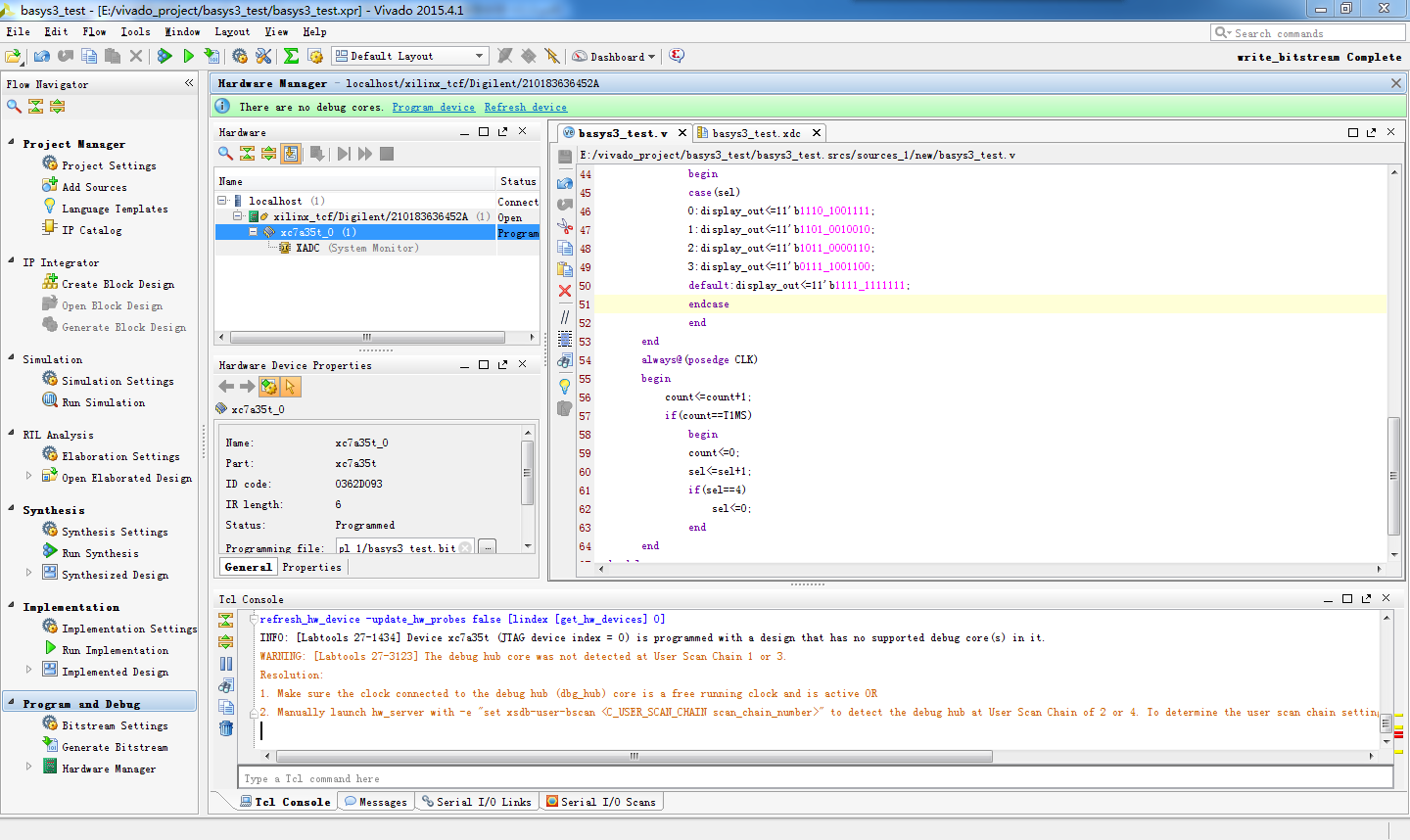
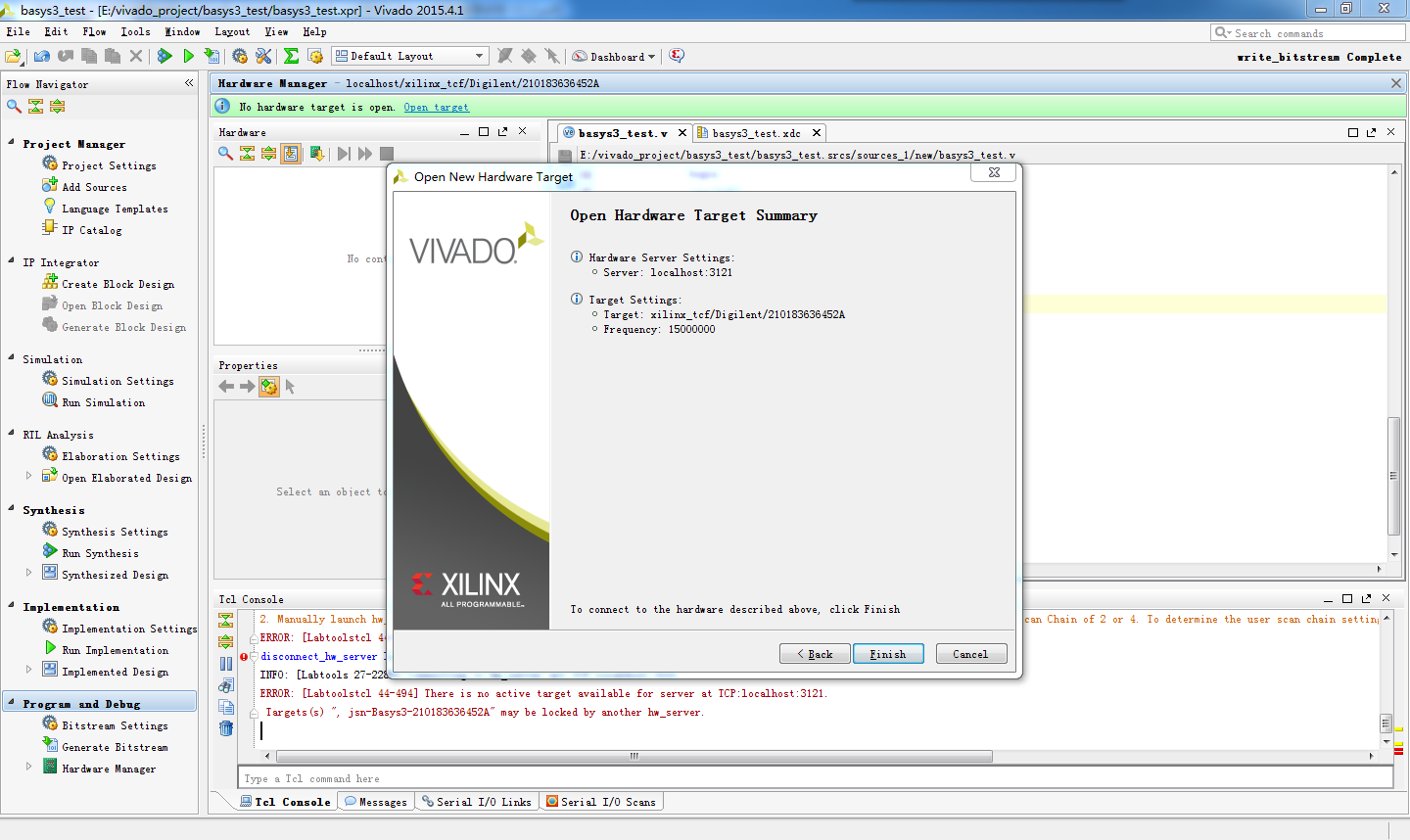
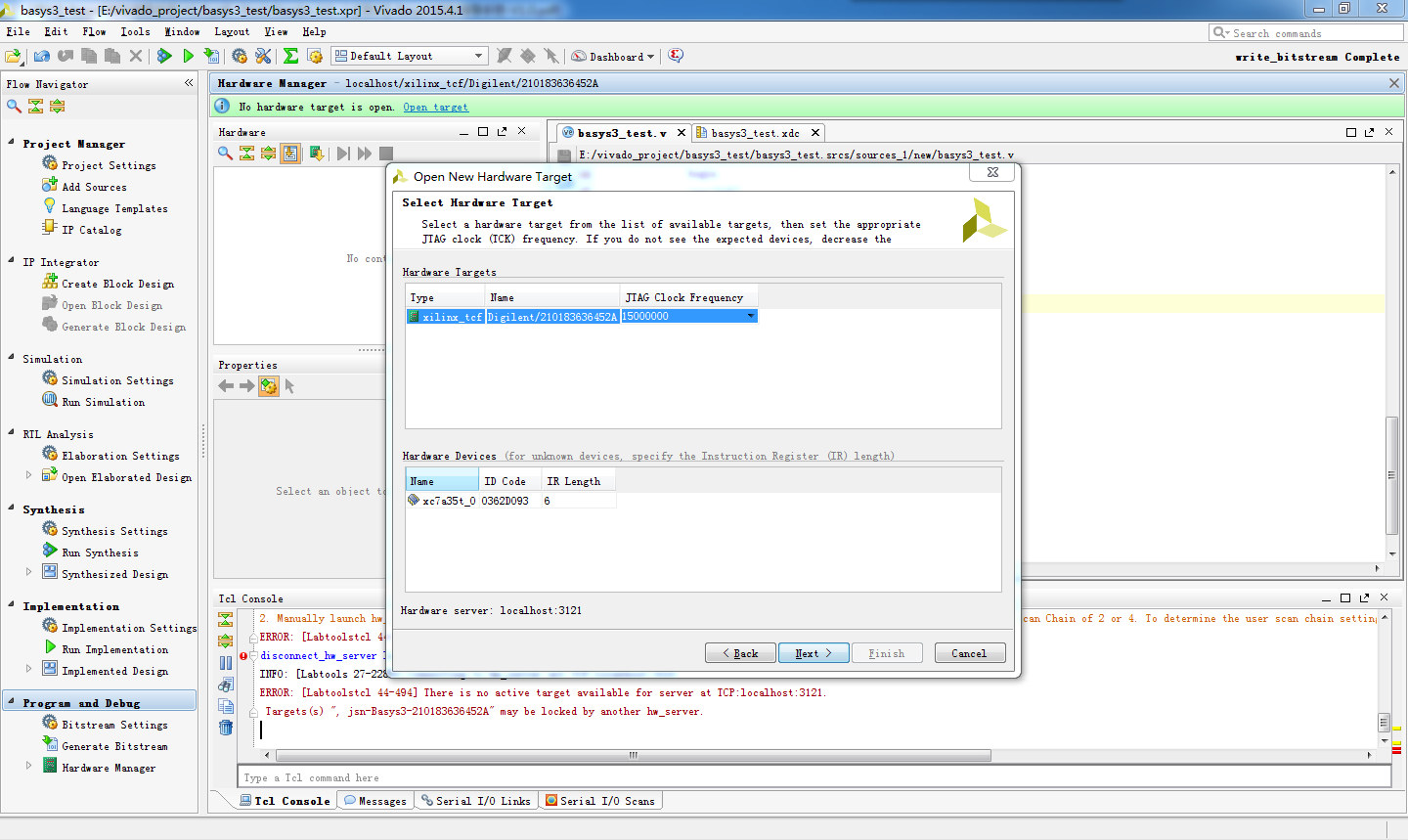
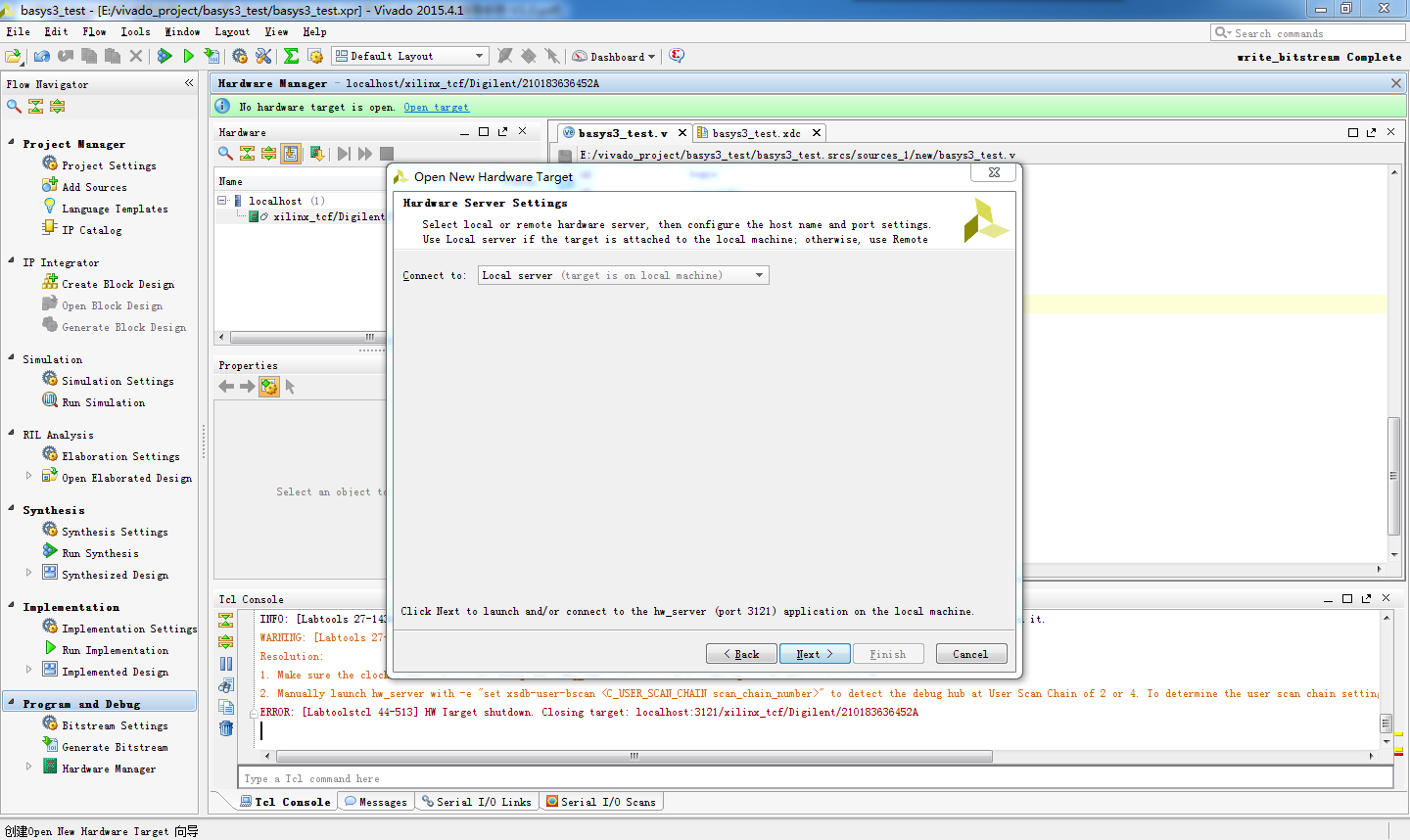
编写完约束文件后点击左侧导航栏中的run implementation进行实现。实现是利用FPGA厂商的实现工具把综合后的逻辑映射到目标器件结构的资源中，决定逻辑的最佳布局，选择逻辑与输入输出功能连接的布线通道进行连线，并产生相应的文件（如配置文件和相关报告）。实现后弹出下框，选择generate bitstream，生成bit流之后选择open hardware manager



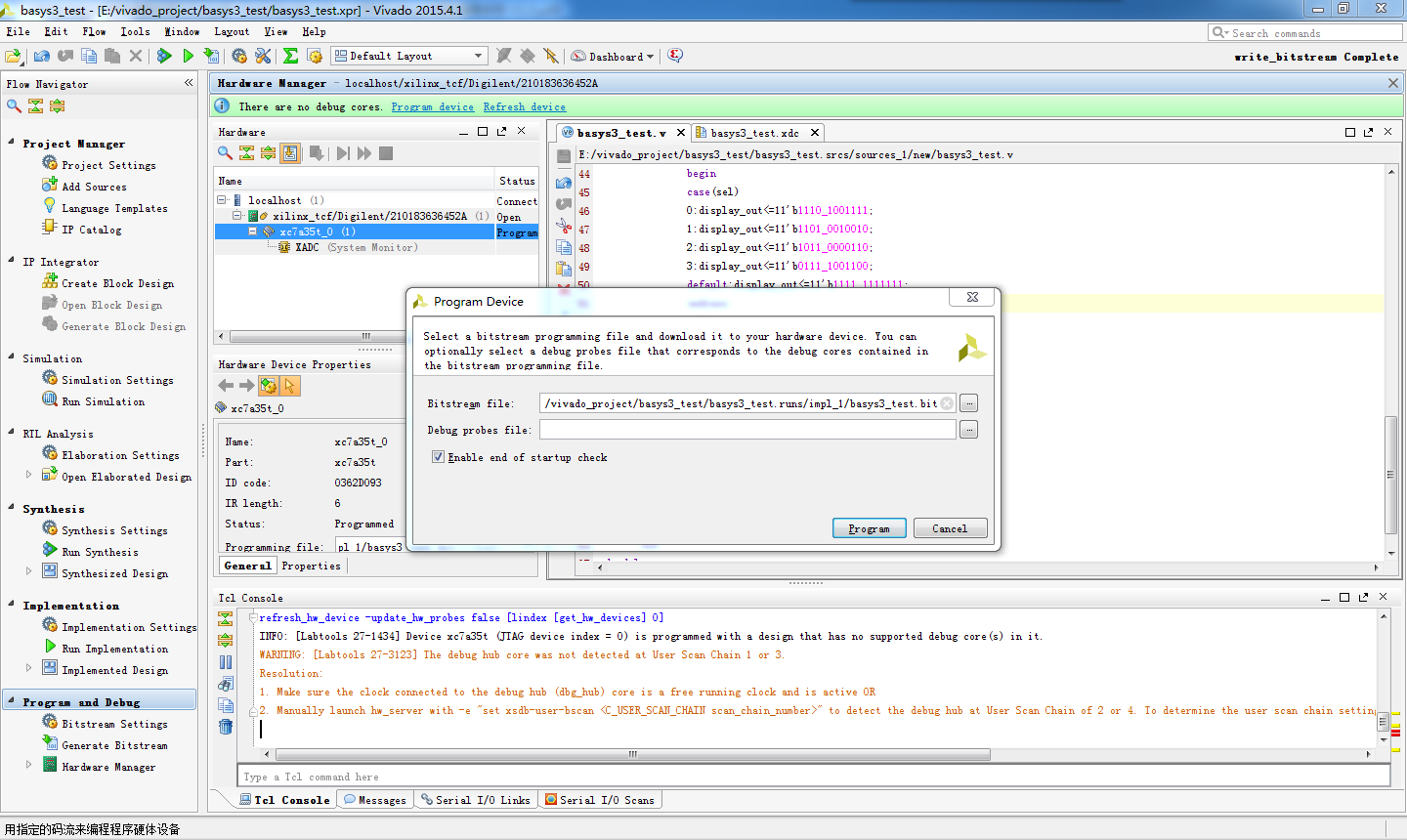
此时，将basys3板子通过usb与电脑连接并上电。点击如图界面中上方的open target，选择open hardware target



点击next，然后可以看到vivado已经识别了basys3，最后finish



在如下界面中右键单击hardware下的芯片，然后选择program device，在弹出的界面中选择刚刚生成的bit流文件，然后点击program即可观看开发板演示。



参考源文件代码

module basys3\_test(

input CLK,

input SW\_in,

output reg[10:0] display\_out

);

reg [19:0]count=0;

reg [2:0] sel=0;

parameter T1MS=50000;

always@(posedge CLK)

begin

if(SW\_in==0)

begin

case(sel)

0:display\_out<=11'b0111\_1001111;

1:display\_out<=11'b1011\_0010010;

2:display\_out<=11'b1101\_0000110;

3:display\_out<=11'b1110\_1001100;

default:display\_out<=11'b1111\_1111111;

endcase

end

else

begin

case(sel)

0:display\_out<=11'b1110\_1001111;

1:display\_out<=11'b1101\_0010010;

2:display\_out<=11'b1011\_0000110;

3:display\_out<=11'b0111\_1001100;

default:display\_out<=11'b1111\_1111111;

endcase

end

end

always@(posedge CLK)

begin

count<=count+1;

if(count==T1MS)

begin

count<=0;

sel<=sel+1;

if(sel==4)

sel<=0;

end

end

endmodule

约束文件中的引脚分配

set\_property PACKAGE\_PIN W5 [get\_ports CLK]

set\_property PACKAGE\_PIN V17 [get\_ports SW\_in]

set\_property IOSTANDARD LVCMOS33 [get\_ports SW\_in]

set\_property IOSTANDARD LVCMOS33 [get\_ports CLK]

set\_property PACKAGE\_PIN W4 [get\_ports {display\_out[10]}]

set\_property PACKAGE\_PIN V4 [get\_ports {display\_out[9]}]

set\_property PACKAGE\_PIN U4 [get\_ports {display\_out[8]}]

set\_property PACKAGE\_PIN U2 [get\_ports {display\_out[7]}]

set\_property PACKAGE\_PIN W7 [get\_ports {display\_out[6]}]

set\_property PACKAGE\_PIN W6 [get\_ports {display\_out[5]}]

set\_property PACKAGE\_PIN U8 [get\_ports {display\_out[4]}]

set\_property PACKAGE\_PIN V8 [get\_ports {display\_out[3]}]

set\_property PACKAGE\_PIN U5 [get\_ports {display\_out[2]}]

set\_property PACKAGE\_PIN V5 [get\_ports {display\_out[1]}]

set\_property PACKAGE\_PIN U7 [get\_ports {display\_out[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[9]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[8]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {display\_out[10]}]

basys3的引脚分配如下图。可知clock的引脚为W5，SW0的引脚为V17，四个七段数码管的引脚分别是U2，U4，V4，W4，W7，W6，U8，V8，U5，V5，U7，V7。依次可以配置上述约束文件。



约束文件

set\_property PACKAGE\_PIN W5 [get\_ports CLK]

set\_property IOSTANDARD LVCMOSS33 [get\_ports CLK]

set\_property PACKAGE\_PIN U16 [get\_ports {LED[0]}]

set\_property PACKAGE\_PIN E19 [get\_ports {LED[1]}]

set\_property IOSTANDARD LVCMOSS33 [get\_ports {LED[0]}]

set\_property IOSTANDARD LVCMOSS33 [get\_ports {LED[1]}]

源文件

module test(

input CLK,

outprt reg[15:0] LED

);

reg[19:0]count =0;

reg[9:0]a=0;

reg[4:0]b=0;

parameter T1MS=50000;

always@(posedge CLK)

begin

if(b==0)

LED<=16’b0000000000000001;

count<=count+1;

if(count==T1MS)

begin

count<=0;

a<=a+1;

if(a==500)

begin

LED<=LED<<1;

b<=b+1;

end

end

if(b==16)

b<=0;

end

endmodule

# UART通讯实验

UART是一种通用串行数据总线，用于异步通信。该总线双向通信，可以实现全双工传输和接收。

首先创建4个源文件，创建方法参考上一个实验。四个源文件实现的功能分别是通过UART进行数据的发送、接收，波特率的选择以及顶层文件的设置。

第一个源文件参考代码，它实现数据发送

1 module uart\_tx

2 (

3 clk,

4 rst\_n,

5 rx\_data,

6 rx\_int,

7 rs232\_tx,

8 clk\_bps,

9 bps\_start

10 );

11

12  input clk;

13  input rst\_n;

14  input clk\_bps;//接收或者发送数据位的中间采样点

15  input [7:0] rx\_data;//接收数据寄存器

16  input rx\_int;//接收数据中断信号，接收到数据期间始终为高电平

17  output rs232\_tx;//RS232发送数据信号

18 output bps\_start;//接收到数据后，波特率时钟启动信号置位

19

20 reg rx\_int0,rx\_int1,rx\_int2;

21 wire neg\_rx\_int;

22

23 always @ (posedge clk or negedge rst\_n)

24 begin

25 if(!rst\_n)

26 begin

27 rx\_int0 <= 1'b0;

28 rx\_int1 <= 1'b0;

29 rx\_int2 <= 1'b0;

30 end

31

32 else

33 begin

34 rx\_int0 <=rx\_int;

35 rx\_int1 <=rx\_int0;

36 rx\_int2 <=rx\_int1;

37 end

38 end

39

40 assign neg\_rx\_int = ~rx\_int1 & rx\_int2;

41

42 reg [7:0] tx\_data;

43 reg bps\_start\_r;

44 reg tx\_en;

45 reg [3:0] num;

46

47 always @ (posedge clk or negedge rst\_n)

48 begin

49 if(!rst\_n)

50 begin

51 bps\_start\_r <= 1'bz;

52 tx\_en <= 1'b0;

53 tx\_data <= 8'd0;

54 end

55

56 else if(neg\_rx\_int)

57 begin

58 bps\_start\_r <= 1'b1;

59 tx\_data <= rx\_data;

60 tx\_en <= 1'b1;

61 end

62

63 else if (num== 4'd11)

64 begin

65 bps\_start\_r <= 1'b0;

66 tx\_en <= 1'b0;

67 end

68

69 end

70

71 assign bps\_start = bps\_start\_r;

72

73 reg rs232\_tx\_r;

74

75 always @ (posedge clk or negedge rst\_n)

76 begin

77 if(!rst\_n)

78 begin

79 num <= 4'd0;

80 rs232\_tx\_r <= 1'b1;

81 end

82

83 else if(tx\_en)

84 begin

85 if(clk\_bps)

86 begin

87 num <= num + 1'b1;

88 case(num)

89 4'd0 : rs232\_tx\_r <= 1'b0;

90 4'd1 : rs232\_tx\_r <= tx\_data[0];

91 4'd2 : rs232\_tx\_r <= tx\_data[1];

92 4'd3 : rs232\_tx\_r <= tx\_data[2];

93 4'd4 : rs232\_tx\_r <= tx\_data[3];

94 4'd5 : rs232\_tx\_r <= tx\_data[4];

95 4'd6 : rs232\_tx\_r <= tx\_data[5];

96 4'd7 : rs232\_tx\_r <= tx\_data[6];

97 4'd8 : rs232\_tx\_r <= tx\_data[7];

98 4'd9 : rs232\_tx\_r <= 1'b1;

99 default: rs232\_tx\_r <= 1'b1;

100 endcase

101 end

102

103 else if (num == 4'd11) num <= 4'd0;

104 end

105 end

106

107 assign rs232\_tx = rs232\_tx\_r;

108

109 endmodule

第二个源文件参考代码，它实现数据接收

1 module uart\_rx   
 2 (  
 3 clk,  
 4 rst\_n,  
 5 rs232\_rx,  
 6 rx\_data,  
 7 rx\_int,  
 8 clk\_bps,  
 9 bps\_start  
 10 )/\*synthesis noprune\*/;  
 11   
 12 input clk;  
 13 input rst\_n;  
 14 input rs232\_rx;//RS232接收数据信号  
 15 input clk\_bps;//接收或者发送数据位的中间采样点  
 16 output bps\_start;//接收到数据后，波特率时钟启动信号置位  
 17 output [7:0] rx\_data;//接收数据寄存器，保存直至下一个数据到来  
 18 output rx\_int;//接收数据中断信号，接收到数据期间始终为高电平  
 19   
 20 reg rs232\_rx0,rs232\_rx1,rs232\_rx2,rs232\_rx3;  
 21 wire neg\_rs232\_rx;//表示数据线接收到下降沿  
 22   
 23 always @ (posedge clk or negedge rst\_n)  
 24 begin  
 25 if(!rst\_n)  
 26 begin  
 27 rs232\_rx0 <= 1'b0;  
 28 rs232\_rx1 <= 1'b0;  
 29 rs232\_rx2 <= 1'b0;  
 30 rs232\_rx3 <= 1'b0;  
 31 end  
 32   
 33 else   
 34 begin  
 35 rs232\_rx0 <= rs232\_rx;  
 36 rs232\_rx1 <= rs232\_rx0;  
 37 rs232\_rx2 <= rs232\_rx1;  
 38 rs232\_rx3 <= rs232\_rx2;  
 39 end  
 40 end  
 41   
 42 assign neg\_rs232\_rx = rs232\_rx3 &rs232\_rx2 & ~rs232\_rx1 & ~rs232\_rx0;  
 43   
 44 reg bps\_start\_r;  
 45 reg [3:0] num;  
 46 reg rx\_int;  
 47   
 48 always @ (posedge clk or negedge rst\_n)  
 49 if(!rst\_n)  
 50 begin  
 51 bps\_start\_r <= 1'bz;  
 52 rx\_int <= 1'b0;  
 53 end  
 54   
 55 else if(neg\_rs232\_rx)  
 56 begin  
 57 bps\_start\_r <= 1'b1;  
 58 rx\_int <= 1'b1;  
 59 end  
 60   
 61 else if(num==4'd12)  
 62 begin  
 63 bps\_start\_r <= 1'b0;  
 64 rx\_int <= 1'b0;  
 65 end  
 66   
 67 assign bps\_start =bps\_start\_r;  
 68   
 69 reg [7:0] rx\_data\_r;  
 70 reg [7:0] rx\_tmp\_data;  
 71   
 72 always @ (posedge clk or negedge rst\_n)  
 73 if(!rst\_n)  
 74 begin  
 75 rx\_tmp\_data <= 8'd0;  
 76 num <= 4'd0;  
 77 rx\_data\_r <= 8'd0;  
 78 end  
 79   
 80 else if(rx\_int)  
 81 begin  
 82 if(clk\_bps)  
 83 begin  
 84 num <= num+1'b1;  
 85 case(num)  
 86 4'd1: rx\_tmp\_data[0] <= rs232\_rx;  
 87 4'd2: rx\_tmp\_data[1] <= rs232\_rx;  
 88 4'd3: rx\_tmp\_data[2] <= rs232\_rx;  
 89 4'd4: rx\_tmp\_data[3] <= rs232\_rx;  
 90 4'd5: rx\_tmp\_data[4] <= rs232\_rx;  
 91 4'd6: rx\_tmp\_data[5] <= rs232\_rx;  
 92 4'd7: rx\_tmp\_data[6] <= rs232\_rx;  
 93 4'd8: rx\_tmp\_data[7] <= rs232\_rx;  
 94 default: ;  
 95 endcase  
 96 end  
 97   
 98 else if(num == 4'd12)  
 99 begin  
100 num <= 4'd0;  
101 rx\_data\_r <= rx\_tmp\_data;  
102 end  
103 end   
104 assign rx\_data = rx\_data\_r;  
105   
106 endmodule

第三个源文件参考代码，它实现波特率的选择

1 module speed\_slect

2 (

3 clk,

4 rst\_n,

5 bps\_start,

6 clk\_bps

7 )/\*synthesis noprune\*/;

8

9 input clk; //50MHZ

10 input rst\_n; //the low level is valid

11 input bps\_start;//接收到数据后，波特率时钟启动信号置位,波特率时钟启动信号

12 output clk\_bps;//接收或者发送数据位的中间采样点

13

14 parameter BPS\_PARA = 5207; //10^6/9600\*1000/20

15 parameter BPS\_PARA\_2 = 2603;

16

17

18 reg clk\_bps\_r;

19 reg [12:0] cnt;

20

21 always @(posedge clk or negedge rst\_n)

22 if(!rst\_n) cnt <= 13'd0;

23 else if ((cnt== BPS\_PARA) || !bps\_start) cnt <= 13'd0;

24 else cnt <= cnt+1'b1;

25

26 always @ (posedge clk or negedge rst\_n)

27 if(!rst\_n) clk\_bps\_r <= 1'b0;

28 else if (cnt == BPS\_PARA\_2) clk\_bps\_r <= 1'b1;

29 else clk\_bps\_r <= 1'b0;

30

31 assign clk\_bps = clk\_bps\_r;

32

33 endmodule

34 /\*

35 parameter bps9600 = 5207, //波特率为9600bps

36 bps19200 = 2603, //波特率为19200bps

37 bps38400 = 1301, //波特率为38400bps

38 bps57600 = 867, //波特率为57600bps

39 bps115200 = 433; //波特率为115200bps

40

41 parameter bps9600\_2 = 2603,

42 bps19200\_2 = 1301,

43 bps38400\_2 = 650,

44 bps57600\_2 = 433,

45 bps115200\_2 = 216;

46 \*/

最后一个源文件参考代码，将其右键，并点击set as top，设置为顶层文件，顶层文件就是将这些小模块综合起来的一个文件。

1 module uart\_top  
 2 (  
 3 clk,  
 4 rst\_n,  
 5 rs232\_rx,  
 6 rs232\_tx  
 7 );  
 8   
 9  input clk;  
10  input rst\_n;  
11  input rs232\_rx;  
12 output rs232\_tx;  
13   
14 wire bps\_start1,bps\_start2;  
15 wire clk\_bps1,clk\_bps2;  
16 wire [7:0] rx\_data/\*synthesis keep\*/;  
17 wire rx\_int;  
18   
19 speed\_slect speed\_rx(  
20 .clk(clk),  
21 .rst\_n(rst\_n),  
22 .bps\_start(bps\_start1),  
23 .clk\_bps(clk\_bps1)  
24 );  
25   
26 uart\_rx uart\_rx(  
27 .clk(clk),  
28 .rst\_n(rst\_n),  
29 .rs232\_rx(rs232\_rx),  
30 .rx\_data(rx\_data),  
31 .rx\_int(rx\_int),  
32 .clk\_bps(clk\_bps1),  
33 .bps\_start(bps\_start1)  
34 );  
35   
36 speed\_slect speed\_tx(  
37 .clk(clk),  
38 .rst\_n(rst\_n),  
39 .bps\_start(bps\_start2),  
40 .clk\_bps(clk\_bps2)  
41 );  
42   
43 uart\_tx uart\_tx(  
44 .clk(clk),  
45 .rst\_n(rst\_n),  
46 .rx\_data(rx\_data),  
47 .rx\_int(rx\_int),  
48 .rs232\_tx(rs232\_tx),  
49 .clk\_bps(clk\_bps2),  
50 .bps\_start(bps\_start2)  
51 );  
52   
53 endmodule

最后是约束文件的创建，用以指定引脚

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -period 10.000 -name sys\_clk\_pin -waveform {0.000 5.000}

-add [get\_ports clk]

set\_property PACKAGE\_PIN U18 [get\_ports rst]

set\_property IOSTANDARD LVCMOS33 [get\_ports rst]

set\_property PACKAGE\_PIN B18 [get\_ports rs232\_rx]

set\_property IOSTANDARD LVCMOS33 [get\_ports rs232\_rx]

set\_property PACKAGE\_PIN A18 [get\_ports rs232\_tx]

set\_property IOSTANDARD LVCMOS33 [get\_ports rs232\_tx]

实验演示参照上一实验。

# 数字频率计实验

数字频率计是采用数字电路制做成的能实现对周期性变化信号频率测量的仪器。频率计主要用于测量正弦波、矩形波、三角波和尖脉冲等周期信号的频率值。其扩展功能可以测量信号的周期和脉冲宽度。

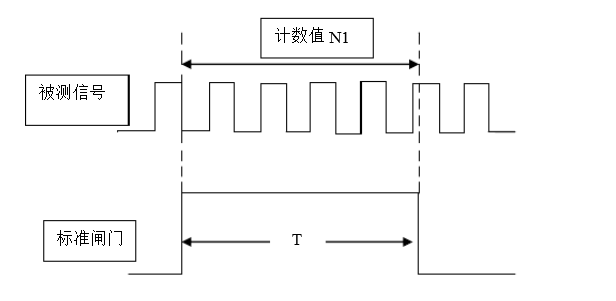
本实验将利用Basys3开发板实现频率测量，并根据测量值的不同选择显示模式。

一般频率计有两种测量方法：

1）计数法

计数法又称测频法，是将被测信号通过一个定时闸门加到计数器进行计数

的方法，如果闸门打开的时间为T，计数器得到的计数值为N1，则被测频率为f=N1/T。改变时间T，则可改变测量频率范围。如图



2）计时法

计时法又称为测周期法，测周期法使用被测信号来控制闸门的开闭，而将标准时基

脉冲通过闸门加到计数器，闸门在外信号的一个周期内打开，这样计数器得到的计数值

就是标准时基脉冲外信号的周期值，然后求周期值的倒数，就得到所测频率值。

计时法适合于对低频信号的测量，而计数法则适合于对较高频信号的测量。但由于用计时法所获得的信号周期数据，还需要求倒数运算才能得到信号频率，而求倒数运算用中小规模数字集成电路较难实现，因此，计时法不适合本实验要求。测频法的测量误差与信号频率成反比，信号频率越低，测量误差就越大，信号频率越高，其误差就越小。但用测频法所获得的测量数据，在闸门时间为一秒时，不需要进行任何换算，计数器所计数据就是信号频率。因此，本实验所用的频率测量方法是测频法。

参考代码如下

module measure\_f(

input clk,

input reset,

input Vi,

output [7:0] led\_8s,

output [3:0] led\_en,

output clk\_out,

output LED\_HZ,

output LED\_KHZ,

output LED\_MHZ

);

reg clk\_out, clk\_pls, clk\_pls2;

reg [26:0] Timer\_count;

reg [27:0] freq\_L, freq\_H;

reg [26:0] freq\_count\_L, freq\_count\_H;

reg [3:0] freq\_out3, freq\_out2, freq\_out1, freq\_out0;

reg dot3, dot2, dot1;

reg HZ, KHZ, MHZ;

reg overflow;

reg [7:0] D3\_buf, D2\_buf, D1\_buf, D0\_buf;

reg [7:0] led\_8s;

reg [3:0] led\_en;

reg [19:0] scanled\_count;

parameter interv=100000;

// reg Vi;

// reg[26:0] tmp\_cnt;

/\*

always@(posedge clk)

begin

if(tmp\_cnt==78)

begin

tmp\_cnt<=0;

Vi<=~Vi;

end

else

tmp\_cnt<=tmp\_cnt+1;

end

\*/

///////////////////////////////////////////////////////// 1s Timer ////////////////////////////////////////////////////////////

always@(posedge clk or posedge reset)

begin

if (reset)

begin

clk\_pls <= 0;

clk\_out <= 0;

Timer\_count <= 0;

end

else

begin

case(Timer\_count)

100000000:begin //100MHz晶振，每过1秒clk\_out翻转一次

clk\_out <= ~clk\_out;

Timer\_count <= 0;

end

0:begin

clk\_pls2 <= 1;

Timer\_count <= Timer\_count + 1;

end

1:begin

clk\_pls2 <= 0;

Timer\_count <= Timer\_count + 1;

end

50000000:begin//clk\_out翻转后延迟0.5s让clk\_pls出一个10ns脉冲.脉冲到来前,前一个计数周期的值用来显示,脉冲沿到来时对前一个计数寄存器清零

Timer\_count <= Timer\_count + 1;

clk\_pls <= 1;

end

50000001:begin

Timer\_count <= Timer\_count + 1;

clk\_pls <= 0;

end

default:Timer\_count <= Timer\_count + 1;

endcase

end

end

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////// freq\_counter ////////////////////////////////////////////////////////////

// assign rst = reset | clk\_pls;

always@(posedge Vi or posedge reset or posedge clk\_pls)

begin

if(reset)

begin

freq\_H <= 0;

freq\_L <= 0;

freq\_count\_H <= 0;

freq\_count\_L <= 0;

end

else if (clk\_pls)

begin

case(clk\_out)

1:begin

freq\_L <= 0;

freq\_count\_L <= 0;

end

0:begin

freq\_H <= 0;

freq\_count\_H <= 0;

end

endcase

end

else if (clk\_out)///////////////////////1s高电平计数///////////////////////////////////////////////////

begin///////////////////////////////////1s高电平计数///////////////////////////////////////////////////

freq\_count\_H <= freq\_count\_H + 1;

if((freq\_H[23:20]==4'b1001) && (freq\_H[19:16]==4'b1001) && (freq\_H[15:12]==4'b1001) && (freq\_H[11:8]==4'b1001) && (freq\_H[7:4]==4'b1001) && (freq\_H[3:0]==4'b1001))

begin

freq\_H[27:24] <= freq\_H[27:24] + 1;

freq\_H[23:20] <= 0;

freq\_H[19:16] <= 0;

freq\_H[15:12] <= 0;

freq\_H[11:8] <= 0;

freq\_H[7:4] <= 0;

freq\_H[3:0] <= 0;

end

else if ((freq\_H[19:16]==4'b1001) && (freq\_H[15:12]==4'b1001) && (freq\_H[11:8]==4'b1001) && (freq\_H[7:4]==4'b1001) && (freq\_H[3:0]==4'b1001))

begin

freq\_H[23:20] <= freq\_H[23:20] + 1;

freq\_H[19:16] <= 0;

freq\_H[15:12] <= 0;

freq\_H[11:8] <= 0;

freq\_H[7:4] <= 0;

freq\_H[3:0] <= 0;

end

else if ((freq\_H[15:12]==4'b1001) && (freq\_H[11:8]==4'b1001) && (freq\_H[7:4]==4'b1001) && (freq\_H[3:0]==4'b1001))

begin

freq\_H[19:16] <= freq\_H[19:16] + 1;

freq\_H[15:12] <= 0;

freq\_H[11:8] <= 0;

freq\_H[7:4] <= 0;

freq\_H[3:0] <= 0;

end

else if ((freq\_H[11:8]==4'b1001) && (freq\_H[7:4]==4'b1001) && (freq\_H[3:0]==4'b1001))

begin

freq\_H[15:12] <= freq\_H[15:12] + 1;

freq\_H[11:8] <= 0;

freq\_H[7:4] <= 0;

freq\_H[3:0] <= 0;

end

else if ((freq\_H[7:4]==4'b1001) && (freq\_H[3:0]==4'b1001))

begin

freq\_H[11:8] <= freq\_H[11:8] + 1;

freq\_H[7:4] <= 0;

freq\_H[3:0] <= 0;

end

else if (freq\_H[3:0]==4'b1001)

begin

freq\_H[7:4] <= freq\_H[7:4] + 1;

freq\_H[3:0] <= 0;

end

else

freq\_H[3:0] <= freq\_H[3:0] + 1;

end/////////////////////////////////////1s高电平计数//////////////////////////////////////////////////

else////////////////////////////////////1s低电平计数//////////////////////////////////////////////////

begin///////////////////////////////////1s低电平计数//////////////////////////////////////////////////

freq\_count\_L <= freq\_count\_L + 1;

if((freq\_L[23:20]==4'b1001) && (freq\_L[19:16]==4'b1001) && (freq\_L[15:12]==4'b1001) && (freq\_L[11:8]==4'b1001) && (freq\_L[7:4]==4'b1001) && (freq\_L[3:0]==4'b1001))

begin

freq\_L[27:24] <= freq\_L[27:24] + 1;

freq\_L[23:20] <= 0;

freq\_L[19:16] <= 0;

freq\_L[15:12] <= 0;

freq\_L[11:8] <= 0;

freq\_L[7:4] <= 0;

freq\_L[3:0] <= 0;

end

else if ((freq\_L[19:16]==4'b1001) && (freq\_L[15:12]==4'b1001) && (freq\_L[11:8]==4'b1001) && (freq\_L[7:4]==4'b1001) && (freq\_L[3:0]==4'b1001))

begin

freq\_L[23:20] <= freq\_L[23:20] + 1;

freq\_L[19:16] <= 0;

freq\_L[15:12] <= 0;

freq\_L[11:8] <= 0;

freq\_L[7:4] <= 0;

freq\_L[3:0] <= 0;

end

else if ((freq\_L[15:12]==4'b1001) && (freq\_L[11:8]==4'b1001) && (freq\_L[7:4]==4'b1001) && (freq\_L[3:0]==4'b1001))

begin

freq\_L[19:16] <= freq\_L[19:16] + 1;

freq\_L[15:12] <= 0;

freq\_L[11:8] <= 0;

freq\_L[7:4] <= 0;

freq\_L[3:0] <= 0;

end

else if ((freq\_L[11:8]==4'b1001) && (freq\_L[7:4]==4'b1001) && (freq\_L[3:0]==4'b1001))

begin

freq\_L[15:12] <= freq\_L[15:12] + 1;

freq\_L[11:8] <= 0;

freq\_L[7:4] <= 0;

freq\_L[3:0] <= 0;

end

else if ((freq\_L[7:4]==4'b1001) && (freq\_L[3:0]==4'b1001))

begin

freq\_L[11:8] <= freq\_L[11:8] + 1;

freq\_L[7:4] <= 0;

freq\_L[3:0] <= 0;

end

else if (freq\_L[3:0]==4'b1001)

begin

freq\_L[7:4] <= freq\_L[7:4] + 1;

freq\_L[3:0] <= 0;

end

else

freq\_L[3:0] <= freq\_L[3:0] + 1;

end/////////////////////////////////////1s低电平计数//////////////////////////////////////////////////

end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////// freq\_unit ////////////////////////////////////////////////////////////

always@(posedge clk\_pls2 or posedge reset)

begin

if(reset)

begin

freq\_out3 <= 0;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

dot3 <= 0;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 0;

MHZ <= 0;

overflow <= 0;

end

else if (clk\_out)///////////////////////use the freq\_counter of the low period//////////////////////////////////////////////////

begin///////////////////////use the freq\_counter of the low period//////////////////////////////////////////////////

if(freq\_count\_L < 1000)

begin

freq\_out3 <= freq\_L[15:12];

freq\_out2 <= freq\_L[11:8];

freq\_out1 <= freq\_L[7:4];

freq\_out0 <= freq\_L[3:0];

dot3 <= 0;

dot2 <= 0;

dot1 <= 0;

HZ <= 1;

KHZ <= 0;

MHZ <= 0;

overflow <= 0;

end

else if(freq\_count\_L < 10000)

begin

freq\_out3 <= freq\_L[15:12];

freq\_out2 <= freq\_L[11:8];

freq\_out1 <= freq\_L[7:4];

freq\_out0 <= freq\_L[3:0];

dot3 <= 1;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 1;

MHZ <= 0;

overflow <= 0;

end

else if(freq\_count\_L < 99995)

begin ///////////////////////////////////////////////////////////////////////

dot3 <= 0;

dot2 <= 1;

dot1 <= 0;

HZ <= 0;

KHZ <= 1;

MHZ <= 0;

overflow <= 0;

if(freq\_L[3:0] < 5)

begin

freq\_out3 <= freq\_L[19:16];

freq\_out2 <= freq\_L[15:12];

freq\_out1 <= freq\_L[11:8];

freq\_out0 <= freq\_L[7:4];

end

else if(freq\_L[7:4] < 9)

begin

freq\_out3 <= freq\_L[19:16];

freq\_out2 <= freq\_L[15:12];

freq\_out1 <= freq\_L[11:8];

freq\_out0 <= freq\_L[7:4] + 1;

end

else if(freq\_L[11:8] < 9)

begin

freq\_out3 <= freq\_L[19:16];

freq\_out2 <= freq\_L[15:12];

freq\_out1 <= freq\_L[11:8] + 1;

freq\_out0 <= 0;

end

else if (freq\_L[15:12] < 9)

begin

freq\_out3 <= freq\_L[19:16];

freq\_out2 <= freq\_L[15:12] + 1;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

else //freq\_L[19:16] < 9

begin

freq\_out3 <= freq\_L[19:16] + 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

end /////////////////////////////////////////////////////////////////////////

else if (freq\_count\_L < 100050) //99995<=f<100050

begin

freq\_out3 <= 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

dot3 <= 0;

dot2 <= 0;

dot1 <= 1;

HZ <= 0;

KHZ <= 1;

MHZ <= 0;

overflow <= 0;

end

else if (freq\_count\_L < 999950)

begin //////////////////////////////////////////////////////////////

dot3 <= 0;

dot2 <= 0;

dot1 <= 1;

HZ <= 0;

KHZ <= 1;

MHZ <= 0;

overflow <= 0;

if(freq\_L[7:4] < 5)

begin

freq\_out3 <= freq\_L[23:20];

freq\_out2 <= freq\_L[19:16];

freq\_out1 <= freq\_L[15:12];

freq\_out0 <= freq\_L[11:8];

end

else if (freq\_L[11:8] < 9)

begin

freq\_out3 <= freq\_L[23:20];

freq\_out2 <= freq\_L[19:16];

freq\_out1 <= freq\_L[15:12];

freq\_out0 <= freq\_L[11:8] + 1;

end

else if (freq\_L[15:12] < 9)

begin

freq\_out3 <= freq\_L[23:20];

freq\_out2 <= freq\_L[19:16];

freq\_out1 <= freq\_L[15:12] + 1;

freq\_out0 <= 0;

end

else if (freq\_L[19:16] < 9)

begin

freq\_out3 <= freq\_L[23:20];

freq\_out2 <= freq\_L[19:16] + 1;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

else //freq\_L[23:20] < 9

begin

freq\_out3 <= freq\_L[23:20] + 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

end /////////////////////////////////////////////////////////////////

else if(freq\_count\_L < 1000500)//999950<=f<1000050

begin

freq\_out3 <= 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

dot3 <= 1;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 0;

MHZ <= 1;

overflow <= 0;

end

else if(freq\_count\_L < 2000000)

begin

dot3 <= 1;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 0;

MHZ <= 1;

overflow <= 0;

if(freq\_L[11:8] < 5)

begin

freq\_out3 <= freq\_L[27:24];

freq\_out2 <= freq\_L[23:20];

freq\_out1 <= freq\_L[19:16];

freq\_out0 <= freq\_L[15:12];

end

else if(freq\_L[15:12] < 9)

begin

freq\_out3 <= freq\_L[27:24];

freq\_out2 <= freq\_L[23:20];

freq\_out1 <= freq\_L[19:16];

freq\_out0 <= freq\_L[15:12] + 1;

end

else if(freq\_L[19:16] < 9)

begin

freq\_out3 <= freq\_L[27:24];

freq\_out2 <= freq\_L[23:20];

freq\_out1 <= freq\_L[19:16] + 1;

freq\_out0 <= 0;

end

else if (freq\_L[23:20] < 9)

begin

freq\_out3 <= freq\_L[27:24];

freq\_out2 <= freq\_L[23:20] + 1;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

else // freq\_L[27:24]<9

begin

freq\_out3 <= freq\_L[27:24] + 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

end

else//1s内所记脉冲数超过2M，指示溢出。

begin

freq\_out3 <= 0;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

dot3 <= 0;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 0;

MHZ <= 0;

overflow <= 1;

end

end///////////////////////use the freq\_counter of the low period//////////////////////////////////////////////////

else ///////////////////////use the freq\_counter of the high period//////////////////////////////////////////////////

begin///////////////////////use the freq\_counter of the high period//////////////////////////////////////////////////

if(freq\_count\_H < 1000)

begin

freq\_out3 <= freq\_H[15:12];

freq\_out2 <= freq\_H[11:8];

freq\_out1 <= freq\_H[7:4];

freq\_out0 <= freq\_H[3:0];

dot3 <= 0;

dot2 <= 0;

dot1 <= 0;

HZ <= 1;

KHZ <= 0;

MHZ <= 0;

overflow <= 0;

end

else if(freq\_count\_H < 10000)

begin

freq\_out3 <= freq\_H[15:12];

freq\_out2 <= freq\_H[11:8];

freq\_out1 <= freq\_H[7:4];

freq\_out0 <= freq\_H[3:0];

dot3 <= 1;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 1;

MHZ <= 0;

overflow <= 0;

end

else if(freq\_count\_H < 99995)

begin ///////////////////////////////////////////////////////////////////////

dot3 <= 0;

dot2 <= 1;

dot1 <= 0;

HZ <= 0;

KHZ <= 1;

MHZ <= 0;

overflow <= 0;

if(freq\_H[3:0] < 5)

begin

freq\_out3 <= freq\_H[19:16];

freq\_out2 <= freq\_H[15:12];

freq\_out1 <= freq\_H[11:8];

freq\_out0 <= freq\_H[7:4];

end

else if(freq\_H[7:4] < 9)

begin

freq\_out3 <= freq\_H[19:16];

freq\_out2 <= freq\_H[15:12];

freq\_out1 <= freq\_H[11:8];

freq\_out0 <= freq\_H[7:4] + 1;

end

else if(freq\_H[11:8] < 9)

begin

freq\_out3 <= freq\_H[19:16];

freq\_out2 <= freq\_H[15:12];

freq\_out1 <= freq\_H[11:8] + 1;

freq\_out0 <= 0;

end

else if (freq\_H[15:12] < 9)

begin

freq\_out3 <= freq\_H[19:16];

freq\_out2 <= freq\_H[15:12] + 1;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

else //freq\_H[19:16] < 9

begin

freq\_out3 <= freq\_H[19:16] + 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

end /////////////////////////////////////////////////////////////////////////

else if (freq\_count\_H < 100050) //99995<=f<100050

begin

freq\_out3 <= 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

dot3 <= 0;

dot2 <= 0;

dot1 <= 1;

HZ <= 0;

KHZ <= 1;

MHZ <= 0;

overflow <= 0;

end

else if (freq\_count\_H < 999950)

begin //////////////////////////////////////////////////////////////

dot3 <= 0;

dot2 <= 0;

dot1 <= 1;

HZ <= 0;

KHZ <= 1;

MHZ <= 0;

overflow <= 0;

if(freq\_H[7:4] < 5)

begin

freq\_out3 <= freq\_H[23:20];

freq\_out2 <= freq\_H[19:16];

freq\_out1 <= freq\_H[15:12];

freq\_out0 <= freq\_H[11:8];

end

else if (freq\_H[11:8] < 9)

begin

freq\_out3 <= freq\_H[23:20];

freq\_out2 <= freq\_H[19:16];

freq\_out1 <= freq\_H[15:12];

freq\_out0 <= freq\_H[11:8] + 1;

end

else if (freq\_H[15:12] < 9)

begin

freq\_out3 <= freq\_H[23:20];

freq\_out2 <= freq\_H[19:16];

freq\_out1 <= freq\_H[15:12] + 1;

freq\_out0 <= 0;

end

else if (freq\_H[19:16] < 9)

begin

freq\_out3 <= freq\_H[23:20];

freq\_out2 <= freq\_H[19:16] + 1;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

else //freq\_H[23:20] < 9

begin

freq\_out3 <= freq\_H[23:20] + 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

end /////////////////////////////////////////////////////////////////

else if(freq\_count\_H < 1000500)//999950<=f<1000050

begin

freq\_out3 <= 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

dot3 <= 1;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 0;

MHZ <= 1;

overflow <= 0;

end

else if(freq\_count\_H < 2000000)

begin

dot3 <= 1;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 0;

MHZ <= 1;

overflow <= 0;

if(freq\_H[11:8] < 5)

begin

freq\_out3 <= freq\_H[27:24];

freq\_out2 <= freq\_H[23:20];

freq\_out1 <= freq\_H[19:16];

freq\_out0 <= freq\_H[15:12];

end

else if(freq\_H[15:12] < 9)

begin

freq\_out3 <= freq\_H[27:24];

freq\_out2 <= freq\_H[23:20];

freq\_out1 <= freq\_H[19:16];

freq\_out0 <= freq\_H[15:12] + 1;

end

else if(freq\_H[19:16] < 9)

begin

freq\_out3 <= freq\_H[27:24];

freq\_out2 <= freq\_H[23:20];

freq\_out1 <= freq\_H[19:16] + 1;

freq\_out0 <= 0;

end

else if (freq\_H[23:20] < 9)

begin

freq\_out3 <= freq\_H[27:24];

freq\_out2 <= freq\_H[23:20] + 1;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

else // freq\_H[27:24]<9

begin

freq\_out3 <= freq\_H[27:24] + 1;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

end

end

else//1s内所记脉冲数超过2M，指示溢出。

begin

freq\_out3 <= 0;

freq\_out2 <= 0;

freq\_out1 <= 0;

freq\_out0 <= 0;

dot3 <= 0;

dot2 <= 0;

dot1 <= 0;

HZ <= 0;

KHZ <= 0;

MHZ <= 0;

overflow <= 1;

end

end///////////////////////use the freq\_counter of the high period//////////////////////////////////////////////////

end

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////// LED Display ////////////////////////////////////////////////////////////////

always@(posedge clk or posedge reset)

begin

if (reset)

D3\_buf <= 8'b11111111;

else

begin

if(dot3)

D3\_buf[0] <= 0;

else

D3\_buf[0] <= 1;

case(freq\_out3)

4'b0000:D3\_buf[7:1]<=7'b1111111;

4'b0001:D3\_buf[7:1]<=7'b1001111;

4'b0010:D3\_buf[7:1]<=7'b0010010;

4'b0011:D3\_buf[7:1]<=7'b0000110;

4'b0100:D3\_buf[7:1]<=7'b1001100;

4'b0101:D3\_buf[7:1]<=7'b0100100;

4'b0110:D3\_buf[7:1]<=7'b0100000;

4'b0111:D3\_buf[7:1]<=7'b0001111;

4'b1000:D3\_buf[7:1]<=7'b0000000;

4'b1001:D3\_buf[7:1]<=7'b0000100;

default:D3\_buf[7:1]<=7'b1111111;

endcase

end

end

always@(posedge clk or posedge reset)

begin

if (reset)

D2\_buf <= 8'b11111111;

else

begin

if(dot2)

D2\_buf[0] <= 0;

else

D2\_buf[0] <= 1;

case(freq\_out2)

4'b0000:begin

if(freq\_out3==4'b0000)

D2\_buf[7:1]<=7'b1111111;

else

D2\_buf[7:1]<=7'b0000001;

end

4'b0001:D2\_buf[7:1]<=7'b1001111;

4'b0010:D2\_buf[7:1]<=7'b0010010;

4'b0011:D2\_buf[7:1]<=7'b0000110;

4'b0100:D2\_buf[7:1]<=7'b1001100;

4'b0101:D2\_buf[7:1]<=7'b0100100;

4'b0110:D2\_buf[7:1]<=7'b0100000;

4'b0111:D2\_buf[7:1]<=7'b0001111;

4'b1000:D2\_buf[7:1]<=7'b0000000;

4'b1001:D2\_buf[7:1]<=7'b0000100;

default:D2\_buf[7:1]<=7'b1111111;

endcase

end

end

always@(posedge clk or posedge reset)

begin

if (reset)

D1\_buf <= 8'b11111111;

else

begin

if(dot1)

D1\_buf[0] <= 0;

else

D1\_buf[0] <= 1;

case(freq\_out1)

4'b0000:begin

if((freq\_out3==4'b0000)&&(freq\_out2==4'b0000))

D1\_buf[7:1]<=7'b1111111;

else

D1\_buf[7:1]<=7'b0000001;

end

4'b0001:D1\_buf[7:1]<=7'b1001111;

4'b0010:D1\_buf[7:1]<=7'b0010010;

4'b0011:D1\_buf[7:1]<=7'b0000110;

4'b0100:D1\_buf[7:1]<=7'b1001100;

4'b0101:D1\_buf[7:1]<=7'b0100100;

4'b0110:D1\_buf[7:1]<=7'b0100000;

4'b0111:D1\_buf[7:1]<=7'b0001111;

4'b1000:D1\_buf[7:1]<=7'b0000000;

4'b1001:D1\_buf[7:1]<=7'b0000100;

default:D1\_buf[7:1]<=7'b1111111;

endcase

end

end

always@(posedge clk or posedge reset)

begin

if (reset)

D0\_buf<=8'b11111111;

else

begin

if(overflow)

D0\_buf<=8'b10010001;

else

begin

case(freq\_out0)

4'b0000:D0\_buf<=8'b00000011;

4'b0001:D0\_buf<=8'b10011111;

4'b0010:D0\_buf<=8'b00100101;

4'b0011:D0\_buf<=8'b00001101;

4'b0100:D0\_buf<=8'b10011001;

4'b0101:D0\_buf<=8'b01001001;

4'b0110:D0\_buf<=8'b01000001;

4'b0111:D0\_buf<=8'b00011111;

4'b1000:D0\_buf<=8'b00000001;

4'b1001:D0\_buf<=8'b00001001;

default:D0\_buf<=8'b11111111;

endcase

end

end

end

always@(posedge clk or posedge reset)

begin

if(reset)

led\_en<=4'b1111;

else

begin

if(scanled\_count==interv)

begin

scanled\_count<=scanled\_count+1;

led\_8s<=D0\_buf;

led\_en<=4'b0111;

end

else if(scanled\_count==interv\*2)

begin

scanled\_count<=scanled\_count+1;

led\_8s<=D1\_buf;

led\_en<=4'b1011;

end

else if(scanled\_count==interv\*3)

begin

scanled\_count<=scanled\_count+1;

led\_8s<=D2\_buf;

led\_en<=4'b1101;

end

else if(scanled\_count==interv\*4)

begin

scanled\_count<=0;

led\_8s<=D3\_buf;

led\_en<=4'b1110;

end

else

scanled\_count<=scanled\_count+1;

end

end

assign LED\_MHZ=(reset)?0:MHZ;

assign LED\_KHZ=(reset)?0:KHZ;

assign LED\_HZ=(reset)?0:HZ;

endmodule

约束文件用以指定引脚

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property PACKAGE\_PIN U17 [get\_ports reset]

set\_property PACKAGE\_PIN T18 [get\_ports Vi]

set\_property PACKAGE\_PIN W7 [get\_ports {led\_8s[7]}]

set\_property PACKAGE\_PIN W6 [get\_ports {led\_8s[6]}]

set\_property PACKAGE\_PIN U8 [get\_ports {led\_8s[5]}]

set\_property PACKAGE\_PIN V8 [get\_ports {led\_8s[4]}]

set\_property PACKAGE\_PIN U5 [get\_ports {led\_8s[3]}]

set\_property PACKAGE\_PIN V5 [get\_ports {led\_8s[2]}]

set\_property PACKAGE\_PIN U7 [get\_ports {led\_8s[1]}]

set\_property PACKAGE\_PIN V7 [get\_ports {led\_8s[0]}]

set\_property PACKAGE\_PIN W4 [get\_ports {led\_en[0]}]

set\_property PACKAGE\_PIN V4 [get\_ports {led\_en[1]}]

set\_property PACKAGE\_PIN U4 [get\_ports {led\_en[2]}]

set\_property PACKAGE\_PIN U2 [get\_ports {led\_en[3]}]

set\_property PACKAGE\_PIN W3 [get\_ports LED\_HZ]

set\_property PACKAGE\_PIN U3 [get\_ports LED\_KHZ]

set\_property PACKAGE\_PIN P3 [get\_ports LED\_MHZ]

set\_property PACKAGE\_PIN L1 [get\_ports clk\_out]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports reset]

set\_property IOSTANDARD LVCMOS33 [get\_ports Vi]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets Vi\_IBUF]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_8s[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_8s[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_8s[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_8s[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_8s[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_8s[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_8s[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_8s[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_en[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_en[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_en[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led\_en[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports LED\_HZ]

set\_property IOSTANDARD LVCMOS33 [get\_ports LED\_KHZ]

set\_property IOSTANDARD LVCMOS33 [get\_ports LED\_MHZ]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk\_out]